

ARGUS BOOKS

WARGAMING ON THE AMSTRAD

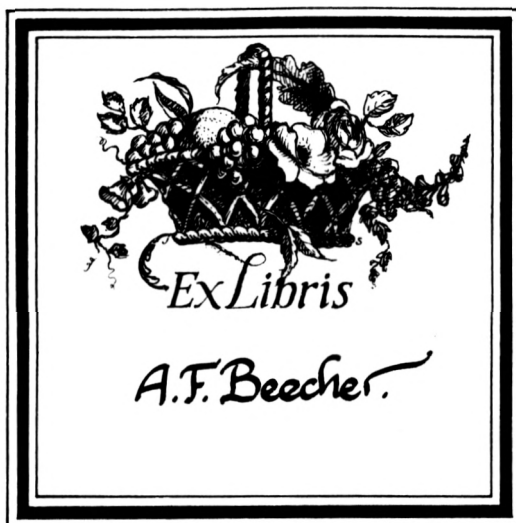
CPC 464,664 & 6128

OWEN & AUDREY BISHOP



Wargaming on the AMSTRAD CPC464, 664 & 6128

**OWEN BISHOP and
AUDREY BISHOP**



ARGUS BOOKS

Argus Books Limited
1 Golden Square
London W1R 3AB
England

© Argus Books Ltd, 1986

ISBN 0 85242 888 X

All rights reserved. No part of this publication may be reproduced in any form, by print, photography, microfilm or any other means without written permission from the publisher.

Phototypesetting by En to En, Tunbridge Wells
Printed and bound by Whitstable Litho

Contents

Preface	
1 Wargaming and computers	1
2 The armies	10
3 The terrain	17
4 Resolving combat	32
5 Handling wargaming data	45
6 JUNGLE ATTACK	55
7 Hidden movement	84
8 Briefing	88
9 NASEBY	94
10 Designing wargames	124
11 Independent action	133
12 OMDURMAN	143
13 Two-computer wargaming	164
14 Modem wargaming	171
Appendix A: Variables and arrays	178
Appendix B: Useful information	183
Index	185

Preface

With this book you can use your computer to play an entirely new type of game. You can, if you wish, use it to teach yourself a new hobby — wargaming. It also allows the experienced wargamer to use the computer to assist at the wargaming table.

Those of you who would like to play the wargames, but not write your own, should omit Chapters 2, 3, 5 and 10 and the final sections of other chapters.

These programs are suitable for the CPC464, CPC664 and CPC6128.

We would like to thank Pace Micro Technology for the loan of their Nightingale Multi Function Modem and their RS232 interface with Commstar software in ROM. We would also like to thank them for their very helpful technical advice.

Wargaming and computers

1

Wargaming, like war itself, originated thousands of years ago. Microcomputers have been with us for less than a decade. In this book we show how the newest technology, incorporated in the microcomputer, can be applied to one of our oldest occupations, the wargame.

Several well-known board games belong to the wargame family. Chess is one such game, though it is much more highly formalised than a wargame. It is generally considered to have been developed several centuries ago in India, where it was known as *chaturanga*. As in modern chess, the pieces represented different kinds of fighting units including elephants, chariots, horses and infantry. In chess, the game involves exploiting the particular attacking and defensive powers of each type of unit, either as individuals or in combination with other units of the same or different kinds. In this respect chess displays its origin as a wargame. However, the wargame concept of different types of terrain over which the units are moved is almost entirely lacking, with its board of 64 identical squares on which only those at the edges and corners can be regarded as being tactically different.

Chess, in which the emphasis is on combat between opposing units, may be contrasted with another ancient and popular offshoot of the wargame family, the Chinese game of *Go*. In *Go* the pieces are identical, and capture is effected not by direct combat, but by encircling a group of enemy pieces. Thus the essence of *Go* is the gaining of enemy territory.

Chess, *Go*, and their variants have retained some of the elements of the wargame, but have lost others. In our opinion, wargames offer to the player all that these games provide and more. The wargame has fighting units of many different types and capabilities. Moreover, these capabilities may alter during the course of the game.

The wargame is fought over a varied terrain, and embodies a degree of realism which makes it a stimulating experience to play. To demonstrate the points we have just made, let us look more closely into the nature of the wargame.

Wargaming

The modern wargame developed in the 18th century, mainly as a technique for use by commanders in planning battles and campaigns, and for training military staff. Wargames are still used for these purposes today, but also are played for interest and enjoyment by an ever-increasing number of enthusiasts among the general public. It is for these enthusiasts and potential enthusiasts that our book is written.

Wargames are usually played between two players, or two groups of players, commanding the opposing sides. Occasionally there may be three or more sides, with a corresponding number of players or groups. Some computer wargames are played by one person, the computer taking control of the opposing side.

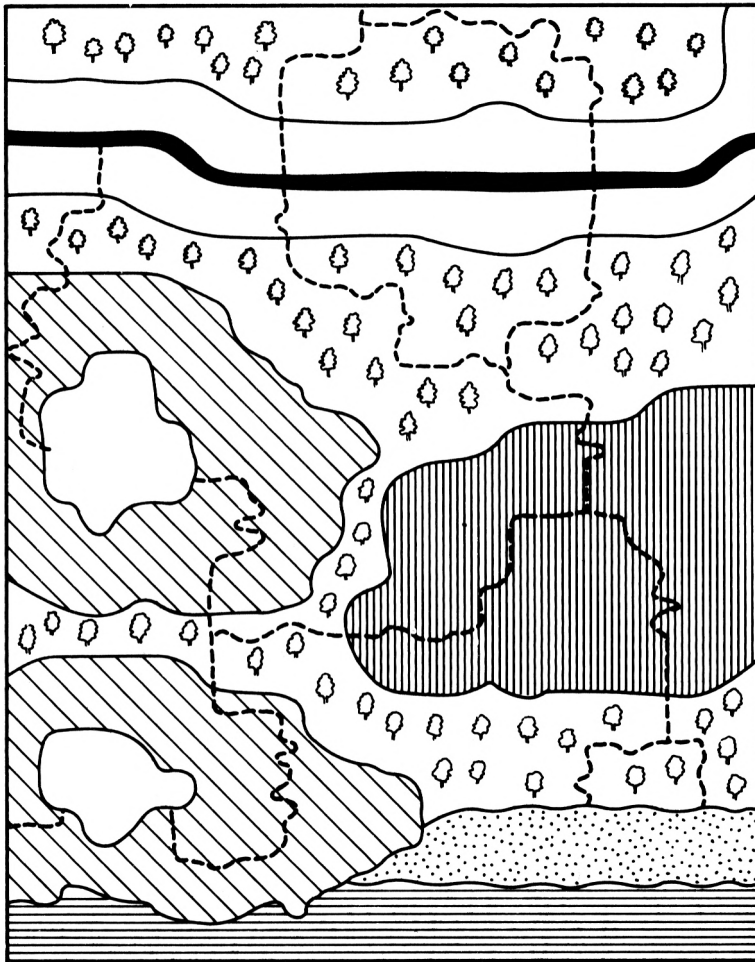
A typical wargame has four main elements: the fighting units, the terrain, the scenario and the rules. The *fighting units*, their numbers, types and combat capabilities are determined by the game designer or maybe by the players themselves. The aim when deciding the nature of the fighting units may be to produce an accurate representation of the armies which took part in an historical battle. Alternatively, the game may be set in a given period but not be based on any real battle. In this case, the composition of the two (or more) opposing sides is arranged so as to result in an interesting, well-balanced game. We go into the subject of fighting units in more detail in Chapter 2.

The *terrain* is the area over which fighting takes place. Unlike the terrain (i.e. the playing boards) of chess or *Go*, the terrain on which a wargame is played is extremely varied (Figure 1.1). It may include natural features such as open ground, woodland, rivers, marshes and mountains as well as artificial features such as roads, railways, towns and minefields. The nature of the terrain affects the rate of movement of each fighting unit. It affects lines of sight. It may provide cover from enemy observers or may provide shelter from enemy fire. As in real warfare, terrain is of overriding tactical importance. We have more to say about terrain in Chapter 3.

The reader will have gathered by now that a wargame designer seeks to produce a model of warfare that is as realistic as possible. Each battle or skirmish therefore has a background story of events leading up to its occurrence. This is the *scenario*. As its name implies, the scenario 'sets the scene' for the game. If the game is based on a real battle, the scenario describes historical events leading up to the battle.

The *rules* of a wargame are designed so that play is a model of what would happen on a real battlefield. Here we usually find it necessary to compromise between realism and playability. For some wargamers, realism is paramount. They play by a complex system of rules with innumerable variations to cover all possible situations. If a situation arises which is not covered by the existing rules, they are prepared to discuss the matter for hours before making the next move. The game moves slowly. Other wargamers prefer to work to a simpler rule system, even though this means that certain aspects of actual warfare are ignored in the game. In this book we have tended to side with the latter group of wargamers.

Rules govern the general plan and conduct of the game, such as the




Terrain

-  Open ground
-  Jungle
-  Steep rocky outcrop
-  Swamp
-  Beach (sandy)
-  Sea

100m

N



 Road

 Track

Fig 1.1 The terrain for a wargame, JUNGLE ATTACK.

number of turns and the sequence of actions in each turn (moving units, firing, rallying units etc). They also cover the details of play such as how far each type of unit may move on each kind of terrain. One other important field covered by the rules is the resolution of combat. In non-computer wargames, players use prepared tables to decide the results of firing at their opponents. The tables take into account the type of weapon, the range, the skill of the firer and the degree to which the target is protected by cover. They also include a random element to represent those factors that ensure that the results of firing at a target are never exactly predictable. This feature is absent from games such as chess in which, if for example, the black bishop is in the position in which it is able to take a white pawn and it is black's turn to play, there is no question about black's ability to move the bishop to the pawn's square and remove the pawn from the board. In a wargame, as in an actual war, having the enemy lined up in the sights of a loaded weapon and pulling the trigger may be an actuality, but the outcome is influenced by many factors, including chance. Consequently, combat resolution, both that for simulating the effects of weapons fired at a distance, or that for simulating hand-to-hand fighting, is a major feature of wargame rules. This topic is discussed further in Chapter 4.

Another feature of wargames' rules is that, except perhaps in competition games, they are open to modification by the players. Players may agree before the game commences, or even during the course of the game, that certain rules be adapted, simplified, ignored or incorporated. This makes it possible to hasten play when time is limited. Or players may adopt rules designed to exploit an interesting tactical situation peculiar to that game. This flexibility is one of the productive features of wargaming.

Methods of wargaming

There are three main methods of conducting a wargame. *Miniature wargaming* (sometimes called model wargaming or table-top wargaming) makes use of model soldiers, vehicles, and artillery moved on a model terrain. *Board wargaming* uses cardboard tokens (Figure 1.2) to represent

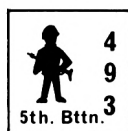


Fig 1.2 Typical counter, as used in board wargaming.

fighting units and armaments, and these are moved on a board which depicts the terrain as a map. *Computer wargaming* makes use of a computer, but there are several different kinds of computer wargame, which we shall discuss later.

The miniature wargamer derives much of the pleasure of the hobby from making or collecting armies of model soldiers accurately representing the troops of a given country and period. They are painted in the correct colours of the times and make an impressive sight on the wargame table.

The modelling of vehicles, and other military equipment, as well as the terrain, is also an absorbing hobby. For some, modelling becomes an end in itself. There are numerous sets of published wargame rules available to cover any period of warfare. Scenarios for wargames are published in books and magazines, or can be designed by the players themselves. The main drawbacks of miniature wargaming (though dedicated miniature wargamers may disagree) are the time taken to set up the game and to make each move, and the large area of table required. Unless one has a special room devoted to the purpose, with a really big table, a miniature wargame is difficult to conduct in the home. Perhaps this is why miniature wargaming is at present the prime occupation of members of wargaming clubs throughout Britain and abroad.

Board wargames are published by a number of firms, many based in the USA. They are sold in boxed sets consisting of map-board, counters, rule-book and often a booklet describing the historical background. Since the game designers aim to cover all possible eventualities in play, the rules tend to be complicated. Frequently the game provides for several scenarios employing the same map and counters. Although board wargames lack the visual appeal of the miniature wargame, they can become highly addictive nevertheless. We know of a board wargame session which lasted from mid-afternoon until the following mid-morning, without a break!

Many of the computer games which are advertised as wargames bear little resemblance to the miniature or board wargames described above. Their topic may be warfare but they owe more to the arcade-style Alien-zapping game than to the true wargame. The true wargamer plans to outwit the enemy by cunning, surprise and superior tactics, not to shoot them down in a frenzy. Such games may be excellent and exciting games in their own right, but they are not wargames in the sense used in this book. Another type of computer wargame is based on strategy, and therefore differs from miniature and board wargames which, are essentially tactical.

Strategy and tactics are distinctive aspects of warfare, though at certain levels the borderline is blurred. *Strategy* involves employing the political and economic resources of a nation to achieve the desired end. If the strategy is successful, the need to fight may never arise! Computer games based on strategy tend to be dominated by numbers – numbers of troops, amounts of money, stocks of food, stocks of weapons and ammunition, and the like. Such a game is ideal for the computer, which is adept at handling numbers but, as far as the player is concerned, the game often consists of trying to out-guess the computer by keying in the ‘correct’ figures. Although strategy is an essential component of warfare, and strategic games can be played enjoyably between human opponents (for example, the game *Diplomacy*), we consider that it makes a less attractive game when played against that mathematical genius, the microcomputer. *Tactics* comprises the planning of military operations and their execution on the battlefield. This is the essence of most miniature and board wargaming. Relatively few computer wargames have been published which resemble the miniature or board wargame in being primarily tactical. Perhaps this is because most computer ‘wargames’ are written by those whose main expertise is in computer programming, not wargaming.

However, there are encouraging signs recently that software publishers are beginning to cater for the tactical wargamer.

Advantages of the micro

In this book we demonstrate how to use the computer to conduct tactical wargames. We show you how to adapt non-computer wargames to the computer, and to design new computer wargames. These games can be played with or without using models, as you prefer. If you are a modelling expert, there is no need to lose the enjoyment of this side of the hobby. Play with your computer alongside the wargaming table. Using the micro allows you more time in which to admire your collection. This book also includes three complete wargames programs. If you are a computer owner new to wargaming, you could begin by keying these in. All three games can be conducted entirely on the computer. Later you can adapt and alter these games, or invent completely new ones, as described in Chapter 9.

Having applied the micro to conventional wargaming, we need not stop at that point. The facilities of the micro can enhance wargaming in several ways. These are listed below among the other advantages of using a computer for wargaming:

- 1 *Instant setting up* The game begins as soon as the program is loaded. Units may be automatically deployed in their initial positions if required.
- 2 *Easier play* By having some of the more complicated rules built into the computer program, the mechanics of playing are simplified, leaving players more time to concentrate on tactics.
- 3 *Facilitates movement of units* Terrain and other factors are taken into account automatically. Only legal moves are allowed.
- 4 *Automatic record keeping* Descriptions of each unit (location, weapons carried, morale, status) are maintained and updated automatically by the computer.
- 5 *Combat resolution* This is more rapidly, more accurately and more realistically achieved by the micro than with conventional combat resolution tables and dice.
- 6 *Interrupted games* The state of the game may be saved to tape or disk at the end of any move. The game is resumed on any future occasion at exactly the same stage at which it was interrupted.
- 7 *Hidden movement* Essential for realism but difficult to achieve satisfactorily without a micro (see Chapter 7).
- 8 *Attrition* Applied automatically (see Chapter 8).
- 9 *Independent action* A further step toward simulating conditions in the field (see Chapter 10).
- 10 *Two-computer wargaming* Two players each control their own forces from two computers (see Chapter 12).
- 11 *Modem wargaming* Solves the problem of the many solo-wargamers who, either because of distance from a wargaming club or for other reasons, are unable to find opponents locally (see Chapter 13).

Wargaming equipment

There is no need for you to obtain any wargaming equipment. But, if you *are* interested in the modelling aspect of the hobby, you will find all you need to know in the magazines and some of the books listed in Appendix B. A visit to your local wargaming club will also provide useful information and contacts.

If you have not tried model wargaming, yet would like to experience it at minimum cost, the book contains cut-out designs of the fighting units required for each game. Copy as many of each of these as the game requires on to thin card, cut them out and colour them. You will need a table on which to play, which should be at least 1.2 m × 0.7 m, but preferably two or three times this size. Cover the table with a cloth (green felt, if possible), under which books or other objects have been placed to provide the necessary relief. A better relief effect can be obtained by cutting suitably shaped sections from polystyrene ceiling tiles. Model shops sell cut-out cardboard buildings, dyed lichen (for use as trees and bushes) and packets of variously coloured granules for scattering on the cloth to indicate different kinds of terrain, such as bare soil, marsh, paths, and roads.

Computer equipment

Of all the popular microcomputers currently available, the Amstrad CPC464, CPC664 and CPC6128 are probably the best for wargaming. The programs in this book have all been extensively tested on the CPC464 and CPC664 and will run on the CPC6128 too. It is preferable to use a colour monitor, for this takes full advantage of the wide range of colours available on Amstrad micros, but if a colour monitor is not available, a monochrome monitor will do.

All programs may be used with tape-based or disk-based systems, though obviously the use of disks speeds loading and saving. However with wargames, which may often take several hours to play, the amount of time spent loading programs and data from tape at the beginning and end of each session of play is negligible. No loading or saving are required during the course of the game itself.

The availability of an inexpensive RS232 serial interface for the Amstrad micros makes it possible to connect two (or more) computers. This equipment is optional but a wargame adapted for play on two computers, as explained in Chapter 12, is an experience quite different from the ordinary one computer game. If you wish to sample this field of wargaming, you will need an Amstrad computer, a serial interface, and a wargaming friend similarly equipped.

The addition of two modems (one for each computer) to this equipment allows you to conduct your games over the British Telecom network with a very moderate increase on your telephone bill.

The wargaming system

This book describes a system for programming computer wargames on Amstrad computers. It is written in Locomotive BASIC 1.0 so as to be

compatible with all three micros, 464, 664 and 6128. Although it has been essential to use machine code in a few instances, we have preferred to use BASIC whenever possible. This allows the reader to modify programs more easily. A complete list defining the variable and array names used in all programs is given in Appendix A.

There are two ways of preparing wargaming programs, which we refer to as the 'Key-in way' and the 'Utility way'. The Key-in way is intended for readers who have little or no experience of programming and merely wish to get the three wargame programs up and running as quickly as possible. Simply key in the listings as instructed in the chapters concerned, and the games are ready to play.

The Utility way is intended for those who wish to design their own games. This way allows the data to be modified so that you can alter the composition of the armies, the nature of the terrain, or the way in which combat is resolved. To do this you need to use the special utility programs described in Chapter 2, 3, and 5. These are the programs which we used when developing the games in this book. Details of how to set up the games are given in the games chapters but below we summarise the essentials.

In the Key-in way you key in the game program listing and save it to tape or disk.

The next item required is a short data program called MCODE. This comprises the code for several machine-code routines used by all the game programs. Once you have typed in and saved this, you use it for all games, so this job has to be done only once. If you are using tape, MCODE must come immediately after the game program. Details of MCODE are in Chapter 3.

Finally, you need a data file. This contains all the details of fighting units, terrain, and any other information relevant to a particular program. In the Key-in way, the data file is prepared by using a *cipher program*. The listing of this is given for each game. Type this in and save it on a separate tape. Before running this program, place the tape with the game program and MCODE on it in the recorder, wound forward to the end of the MCODE program. When the cipher program is run, it creates the required data file and saves it on the tape immediately after MCODE.

You now have three items on the same tape, in the following order: the game program, MCODE, and the data file. When the game program is run it automatically loads the machine code and data file, and then commences the game.

The description above applies to tape users. With a disk system, all items must be on the same disk. Type in and save the game program, save MCODE, type in and save the cipher program (this need not be on a separate disk), and run the cipher program to create the data file. To play, run the game program.

You may be wondering why it is recommended to save the cipher program, even though it is used only once for preparing the data file. The reason is that you may make mistakes when typing the cipher program. Such errors will appear in the data file and will probably upset the working of the game. In particular, details of fighting units may be incorrect or the

terrain map may have faults. It is relatively easy to re-load the cipher program, list it and look for mistakes. When these have been corrected, the program is run again to create a new corrected data file.

The Utility way differs from the above in that the cipher program is not used. Instead, you create separate data files for each army (using the utility program RECRUITER, Chapter 2), for the terrain map (using CARTOGRAPHER, Chapter 3), and various data tables (using TABLER, Chapter 5). At this stage you can modify the entries, so as to produce armies and terrain differing from those in the game descriptions, according to your preferences. The various files are then amalgamated into one composite data file for the game, using INTELLIGENCE (Chapter 5). This is then equivalent to the data file created by a cipher program (see Key-in way). The intelligence utility allows additional flexibility in game design, for you can bring together different pairs of armies on different terrains to generate new wargames while using the original wargame program. The Utility programs are also used when you are inventing your own totally new wargames. When you key in the first game listing (preferably JUNGLE ATTACK, Chapter 6, as this is the easiest one to begin with) you will find that the listing is rather a long one. But, for the second and third games, you will have very much less keying-in to do. This is because each game listing consists of two parts. The first part, which is the main game program, is short (only about 20 lines long) and is different for each game. The remainder of the listing, consists of standard subroutines which are used in all or almost all of the games. Once you have keyed in one game listing you can quickly convert the listing to another game. All you do is load the existing program, replace the main game section with that of the new game, and then add the few extra subroutines (if any) needed for the new game.

The next step

If you intend to adopt the Key-in way, turn to the end of Chapter 3 to key in MCODE, then to Chapter 6 to key in and play JUNGLE ATTACK. There is no need to go deeply into the other intervening chapters, though if you read just the introductory sections of these chapters it will give you a better understanding of how the games work.

If you propose to adopt the Utilities way, turn to the next chapter.

2 The Armies

In most wargames there are two opposing armies, though in some there may be more. Occasionally one of the 'armies' may be the civilian population of the area, who may be neutral or may give armed support to one side or the other. They may even change sides during the game.

Each army consists of a number of units. These are usually fighting units, though occasionally they may be non-combatant units such as baggage trains. They may sometimes be vehicles or military equipment, such as artillery. Depending on the scale of the game, a unit may represent any number of fighters from a single soldier to a whole army corps of several thousands. For example, in a skirmish wargame, such as JUNGLE ATTACK (Chapter 6), each unit represents one man. In a battle wargame such as NASEBY (Chapter 9) different units represent different numbers of men, ranging from 500 to 1500. The numbers depend partly on how many fight under each sub-commander in the actual battle, and partly on the requirements of the game design. If there are many units, each move takes too long to play and the display on the screen becomes difficult to interpret. Yet if there are too few units, the tactical possibilities open to the players become limited.

It is rare in wargaming for all units to be identical. Usually there are several different types of units in each army, each type of unit having its own characteristics. Armies are distinguished on the screen by symbols representing each unit, displayed in the distinctive colours of each army. Different types of unit are distinguished by special symbols.

The characteristics of each unit, or each type of unit, include the number of soldiers comprising the unit and features such as the weapons carried, their skill at using these weapons, skills at hand-to-hand fighting, their morale and their status. Morale is an important factor. If morale is high, the attack is pressed with greater vigour. If it is low, a unit may surrender or desert. Within the general term 'status' we include a number of features which may vary from game to game. Status nearly always includes reference to the fighting condition of the unit. For instance, the status of a unit representing a single soldier may be 'active', 'wounded', 'killed in action' or 'panicked'. Until a unit is 'killed in action' the status may change several times during the game. Panic is a mental state which temporarily prevents the soldier from taking an effective part in the battle. A panicked

soldier may become completely inactive or, worse, may fire at units on his own side!

If the units represent more than one soldier, the size of the unit will decrease as a result of combat as members of the group are severely wounded or killed. The number of men is a feature of each unit and the computer reduces this number after each engagement. If the number falls below a certain level, the unit may surrender — it is rare in warfare for a unit to continue fighting when its strength has been severely reduced. In some games, the rules may allow remnants of units to be reformed into a single larger unit before this danger-point is reached.

Another feature of each unit is its location on the terrain. The map of the terrain is subdivided into rows and columns of squares (Figure 2.1). The

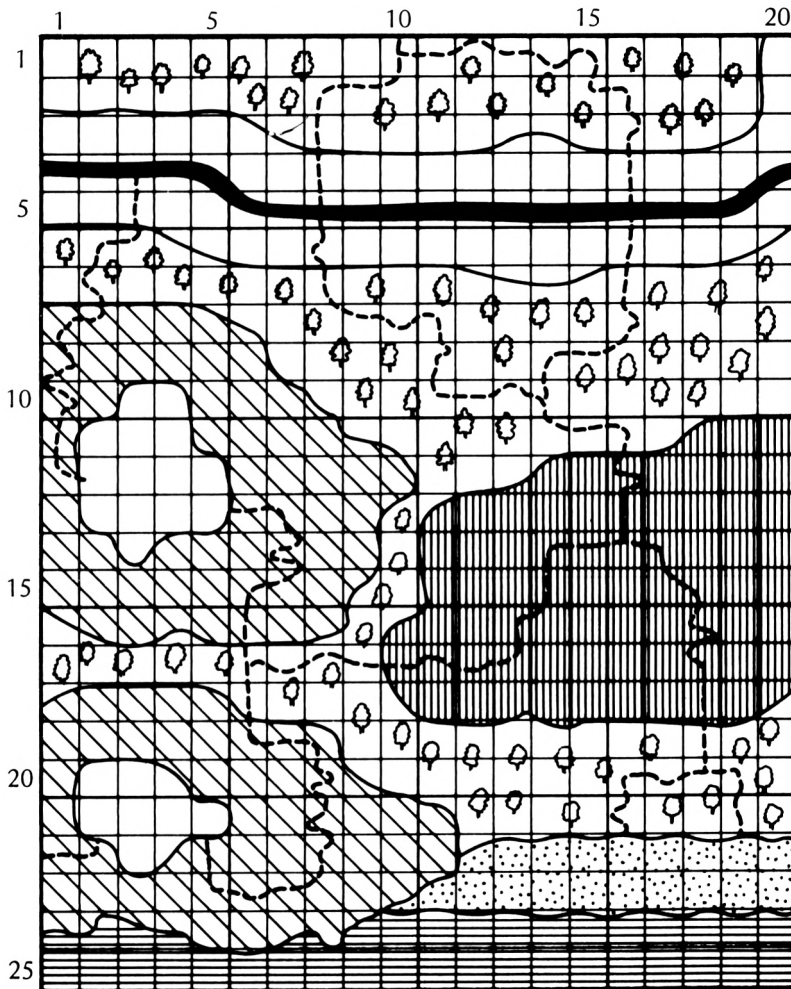


Fig 2.1 Map design for JUNGLE ATTACK.

location of the unit is defined as the row and column of the square it is in. Depending on the scale of the game and the size of the units, it may be allowable for several units to be located in the same square. In historically based games, the location of each unit at the beginning of the game is often decided by how the units were deployed in the real battle. In other games we give the players the opportunity of placing (or deploying) the units as they see fit, subject perhaps to certain conditions set out in the rules.

Recruiter

This utility program allows you to specify the types of unit in an army and how many there shall be of each type. It also allows you to specify the features of each unit as they are to be at the start of the game.

Key in and save the program listed below. Note especially that the variable on lines 40, 50 and elsewhere is a small letter 'el', not a figure 'one'. If you do not intend to use a printer, you may omit lines 690–900, inclusive. Before running the program you need a table which lists the composition of the army. For the games in this book, such tables are in Appendix D.

```

10 REM ** RECRUITER **
20 CLEAR:CLS:INPUT "Army name";t$:length=8
30 GOSUB 960:army$=t$
40 l=0:u=15:m$="Paper number":GOSUB 930 :pa=number

50 l=0:u=15:m$="Pen number":GOSUB 930 :pe=number
60 IF pe=pa GOTO 40
70 l=1:u=255:m$="Types of unit":GOSUB 930
80 types=number:DIM types$(types),units(types)
90 CLS:FOR k=1 TO types
100 PRINT "Name of unit type No."k";:INPUT types$:leng
th=6:t$=types$
110 GOSUB 960:types$(k)=t$:NEXT
120 PRINT"Is all correct? Y/N":GOSUB 1000
130 IF a$="N" THEN 90
140 CLS:FOR k=1 TO types:PRINT"How many "types$(k)" (
type "k")";
150 INPUT nunit$
160 nunit=VAL(nunit$):IF nunit<0 THEN 140
170 units(k)=nunit:units=units+nunit:NEXT
180 CLS:PRINT"Specified starting position? Y/N":GOSUB
1000
190 IF a$="Y" THEN sp=-1
200 l=0:u=5:m$="How many details":GOSUB 930
210 ndetails=number:IF ndetails=0 THEN 260
220 CLS:PRINT"Descriptions of details"
230 DIM rnames$(5):FOR k=1 TO ndetails
240 PRINT"Detail"k";:INPUT rnames$:length=6:t$=rnames$
250 GOSUB 960:rnames$(k)=t$:NEXT
260 CLS:PRINT"Description so far:-
270 PRINT"Army name: "army$
280 PRINT"Paper number = "pa:PRINT"Pen number = "pe
290 FOR k=1 TO types:PRINT types$(k)" "units(k):NEXT
300 PRINT"Total units:"units
310 IF sp THEN PRINT"Specified starting positions"

320 IF ndetails=0 THEN 340:ELSE PRINT"Details for:"
330 FOR K=1 TO ndetails:PRINT rnames$(k):NEXT
340 PRINT"Is all correct? Y/N":GOSUB 1000
350 IF a$="N" THEN 20

```

```

360 DIM sym(units)
370 IF sp THEN DIM rsp(units),csp(units)
380 IF ndetails>0 THEN DIM details(units,ndetails)
390 FOR k=1 TO units
400 count=1:cum=0:WHILE cum<k
410 cum=cum+units(count)
420 count=count+1:WEND
430 CLS:PRINT "Unit No."k"      Type "types$(count-1)

440 l=0:u=255:m$="Symbol number":GOSUB 930:sym(k)=num
ber
450 IF ndetails=0 THEN 480
460 FOR d=1 TO ndetails
470 PRINT rnames$(d);"="  ";:INPUT details(k,d):NEXT

480 IF NOT sp THEN 510
490 INPUT "Starting row";rsp(k)
500 INPUT "Starting column";csp(k)
510 PRINT"Is all correct? Y/N":GOSUB 1000
520 IF a$="N" THEN 430
530 NEXT
540 CLS:PRINT"Army list requires"11+9*units;"bytes"

550 file$=army$
560 OPENOUT file$
570 PRINT#9,types,units,ndetails,sp,pa,pe
580 FOR k=1 TO types:PRINT#9,units(k):NEXT
590 FOR k=1 TO units:PRINT#9,sym(k):NEXT
600 IF NOT sp THEN 630
610 FOR k=1 TO units:PRINT#9,rsp(k):NEXT
620 FOR k=1 TO units:PRINT#9,csp(k):NEXT
630 IF ndetails=0 THEN 660
640 FOR k=1 TO units:FOR d=1 TO ndetails
650 PRINT#9,details(k,d):NEXT:NEXT
660 CLOSEOUT
670 PRINT"Backup? Y/N":GOSUB 1000
680 IF a$="Y" THEN 560
690 PRINT"Printout? Y/N":GOSUB 1000
700 IF a$="N" THEN CLS:PRINT"RECRUITER finished":END
710 PRINT#8,"Army list: "army$
720 PRINT#8,"Paper number is "pa
730 PRINT#8,"Pen number is "pe
740 PRINT#8,"No. of types of unit: ";types
750 PRINT#8,"Types:"
760 firstu=1:FOR k=1 TO type:PRINT#8, types$(k);" ";u
nits(k);:IF units(k)>0 THEN PRINT #8, " Units (";:ELS
E PRINT #8: GOTO 780
770 PRINT#8, firstu;" TO ";firstu+units(k)-1;")":firs
tu=firstu+units(k)
780 NEXT
790 PRINT#8,"Total "units" units
800 PRINT#8,"Details are:"
810 PRINT#8,"Unit Type Symbol Row Col ";rn
ames$(1);" ";rnames$(2);" ";rnames$(3);" ";rnames
$(4);" ";rnames$(5)
820 FOR k=1 TO units
830 count=1:cum=0:WHILE cum<k
840 cum=cum+units(count)
850 count=count+1:WEND
860 IF k<10 THEN PRINT#8," ";
870 PRINT#8,k;SPACE$(4)types$(count-1)sym(k);SPACE$(4
);:IF sp THEN PRINT#8,rsp(k)SPACE$(3)csp(k);:ELSE PRI
NT#8,0 SPACE$(3)0;
880 IF ndetails>0 THEN FOR d=1 TO ndetails:PRINT#8,SP
ACE$(5)details(k,d);:NEXT:PRINT#8:NEXT
890 PRINT"Repeat printout? Y/N":GOSUB 1000
900 IF a$="Y" THEN 710
910 CLS:PRINT"RECRUITER finished"
920 END

```

```

930 number=-1000:WHILE number<1 OR number>u:REM SELEC
TNUMBER ***
940 PRINT:PRINT m$ (";1;" to ";u;");
950 INPUT number$:number=VAL(number$):WEND:RETURN
960 text1=LEN(t$):REM FORMSTRING ***
970 FOR j=1 TO (length-text1)
980 t$=t$+" ":NEXT
990 t$=LEFT$(t$,length):RETURN
1000 a$=UPPER$(INKEY$):REM Y/N ***
1010 IF a$<>"Y" AND a$<>"N" THEN 1000
1020 RETURN

```

Using recruiter

Run the program. Place the tape or disk, on which you wish the army description to be saved, in the recorder or disk drive. Below we list the questions you are asked at each stage of the program, and suggest how you should reply.

Army name This name may be up to eight characters long. The syntax of the messages in the games programs assumes that the name is in the form of an adjective, for example "Allied", "British", "Charles", "Japanese", "Aussie" (or "Australn"). Note that the description will be saved with this army name as the file-name. Make sure that you have not already saved an army description with the same name on the same tape or disk.

Paper colour/Pen colour The colours are used for the symbols representing the units. In these games we keep to the Mode 0 default colours for the pen and paper, except for paper/pen 14, which is redefined as white instead of flashing blue/yellow (see Chapter 3). Choose contrasting colours for the pen and paper. The paper colour should preferably not be one which is used for large areas of the terrain map.

Types of unit There can be up to 255 types of unit in the army, though usually the number required is far less than this.

Name of unit type These names are used only for identifying the different types during the course of this program, to help you type the details in correctly. They are not saved to tape or disk. They can be up to six characters long. Unless you are taking a print-out of the army description later in the program, remember to note down the name of each numbered type for future reference. At this point you are asked to confirm that the type names are correct. If so, press key 'Y' and the program continues. If not, press 'N' and key them in again.

How many of each type The names of each type are displayed in order and you are asked to key in how many there are to be of each type.

Specified starting position Type 'Y' if the initial row and column of any one or more of the units is to be specified. If players are to be allowed freedom to deploy units anywhere, subject to the rules of the game, key 'N'.

How many details? There may be up to five details, which include features such as morale, weapon skill, arms carried, and status (see above). When working out the number of details required, take into account the conventions used in many of the subroutines in the games programs as follows:

Detail 1 is the *status* detail (see above); this detail is almost always required.

Detail 2 is the *arms* detail. This records what weapons are being carried, how many grenades are carried, and the like. The exact way in which the arms are coded as a number vary from game to game (see Chapter 5).

Detail 3 is the *weapon skill* detail, usually a rating between 0 and 9.

Detail 4 is the *combat potential* detail in those games involving individual soldiers. In games in which a unit represents more than one man it is the *unit size* detail.

Detail 5 is the *morale* detail.

Not all of the five details may be relevant to a given game but, where they are relevant, it is important to adhere to the scheme above. Otherwise, the subroutines handling these details will not work. Alternatively, the subroutines may be amended or new subroutines written to operate with a revised scheme of details. This could include details relating to features of combat other than those listed above. Assuming that you have decided to keep to the conventional scheme above, if the game depends on morale, you need five details. Even if combat potential does not enter into that game, in which case Detail 4 is a *null* detail, you still need five details for another example, if a game depends on weapons skill but not on combat potential or morale, you require only three details.

Descriptions of details Key in a name (maximum length six characters) to identify each detail, e.g. 'Status', 'Wpnskl', 'Morale', or 'Null'. These names are not saved at the end of the program. They are used to help you while entering the values for each detail. The names appear on the print-out later.

At this stage you are given a summary of all that you have entered so far and asked if all is correct. Key 'Y' if all is correct. If you key 'N', you are taken back to the beginning of the program.

Unit details You are now asked to key in the specification of each unit in turn. The display shows the unit number and what type of unit it is. Refer to your army table when entering the specification, as described below.

Symbol number When the game is played, each unit is represented by a graphics symbol, which is normally a user-defined one. The design of each symbol to be used is contained in the SYMBOLS subroutine of the game program. The range of user defined symbols allocated for units is restricted to ASCII 234 to 255. This allows up to 22 different symbols to be used, enough for 22 different types of unit. Equivalent units on opposing sides may use the *same* symbol, since they are distinguished from each other by being in different colours. If the game requires more than 22 types of unit you may use any of the alphanumeric characters and existing graphics characters as unit symbols, including the control code symbols (see Figure 3.2, Chapter 3).

Details Unless you have previously indicated that there are to be no details, each detail is named in turn. Key in the appropriate value from your army table. Type '0' for any detail that is not relevant to the game (a *null* detail).

Starting row/Starting Column The request for this information appears only if you stated earlier that starting positions are to be specified. Key in

the row and column numbers. In some games, certain units are deployed at will by the players. Enter zero for their row and column. Sometimes certain units are not to be available at the start of the game, but are to be deployed either automatically or by the players at a later stage. Such units could be reinforcements which arrive at the battle area after the fighting has commenced. For units such as these, enter zero for row and column. Units with zero row or column are not displayed on the terrain map.

All correct? This question applies to the particular unit for which you have just completed the specification. Inspect what you have typed carefully and, if all is correct, key 'Y', and the computer then asks for the description of the next unit. If you key 'N', you are able to enter the description for that unit again.

Army list bytes When all unit descriptions are complete the computer informs you how many bytes of memory are needed to store all the information you have entered. The amount of memory allocated to army data is generous (see Chapter 5) so there is little need to be concerned with this figure unless the armies used in the game have more than 100 units in total.

Saving The description is saved automatically to disk. With tape, you are asked to press RECORD and PLAY, then any key. You are given the opportunity to take as many back-up copies as you wish.

Print-out Finally you are able to print out as many hard copies of the army list as you require. The program then ends.

Program description

20–130 Entering main descriptors for the army — name, colours, names of types.

140–170 Number of units of each type.

180–190 Whether starting positions are to be specified.

200–250 Names of details.

260–350 Display of description, and request to verify.

360–380 Dimensioning arrays.

390–530 Describing each unit and verifying description.

540 Number of bytes required.

550–680 Saving, with back-up.

690–920 Print-out: end of program.

930–950 Subroutine to accept a numeric input, within given limits.

960–1020 Subroutine to truncate or pad a string to a prescribed length.

The Terrain

3

The terrain on which the conflict takes place is often a major feature of a wargame. The main aim of an army may be to establish a bridgehead across an unfordable river, or to dislodge a contingent of the enemy who are well established on the summit of a hill. Marshy areas impede the advance of cavalry. Woodland provides the advantage of cover but slows down the rate of an advance. The nature of the terrain needs to be taken into account at almost all stages of the game. In miniature wargaming, the terrain is modelled on the wargame table to a suitable scale. If the area is a hilly one and the slope of the land is important for tactical reasons, the model is usually contoured. In board wargames and computer wargames we use a map, drawn to scale. Contrastingly coloured areas and various symbols are used to denote the several types of terrain within the area, to show the location of features such as roads and buildings, and perhaps to indicate the slope of the ground.

The map display used in the games in this book is based on rows and columns of 'squares' (see Figure 2.1). For a battle ranging over a wide area, the map can have up to 1000 squares. There may be up to 40 columns of squares. The maximum number of rows allowed depends on the number of columns selected. A map with about 30 columns and rows is sufficient for most games. Even a map of this size is too large to be displayed on the screen in its entirety. At any time, the screen shows a section of the map, 20 columns wide by 11 rows. This, incidentally, is the minimum size that the map may have.

Wargame scales

When we try to represent warfare as a wargame, we have three scales to consider:

- The unit scale (or figure scale)

- The time scale

- The map scale (or table-top scale)

We are discussing the question of scales at this point since they must be fixed before the map is prepared. The scales for the wargames in this book

have been decided on already but, when you are designing your own wargames, you need to consider the points mentioned below.

All three scales are interrelated. In theory, their relationship can be specified exactly, but we may need to compromise on this aspect and make some approximations in order to arrive at a workable game design.

The *unit scale* determines how many soldiers are represented by one unit (model soldier or map symbol). For a skirmish with miniatures, the scale *1 model soldier = 1 soldier* is the only one that provides realism. This scale is nearly always practicable since relatively few men are involved in a skirmish. The same consideration applies for skirmish wargames on the computer. For battles in which tens of thousands of warriors are in conflict, this scale is out of the question. Even with a few hundred men taking part, we may run into difficulties. We therefore let one model soldier, or one unit symbol on the map, represent more than one soldier. In the commonly used unit scales, one unit represents 5, 10, 20, 50, 100 or even several thousand men. The scale need not be the same for all units in the game. A frequent example is when one unit represents the commander of the army while other units may represent several hundred infantry men.

In historical games it may be desirable for the scale to vary over a limited range, since not all regiments or other units taking part in a battle would have been made up of the same number of men.

The *time scale* is the length of real time corresponding to one turn or one phase of the game. In a skirmish, action is brisk and the scale *1 turn = 1 minute of combat time* is often used. However, adopting this scale does not prevent certain wargamers from taking an hour to play each turn! In a battle game, the scales *1 turn = 5 minutes*, or *1 turn = 1 hour*, may be more reasonable, depending on the actual length of the battle and the rate at which critical stages of the battle developed. In a campaign game, a turn might be equivalent to a day or even a week. In short, the time scale is selected to divide the whole action, be it a skirmish or a season's campaign into a reasonable and playable number of stages, the *turns*.

The *map scale* of a computer wargame is the distance represented by the length of the side of a 'square'. Therefore the distance a unit moves is reckoned in 'squares'. Distances moved must be related to the distances a real unit would move on real terrain in the real time corresponding to one turn. In designing the game, we have to ensure that a normally active unit is able to move several maps squares each turn. The number must not be too great, or we shall need a map with a prohibitively large number of rows and columns. On the other hand, the number must not be too small or it will not be possible to arrange for differential rates of movement for different types of unit, or for units moving over various types of terrain. The following example shows how a suitable number may be derived. The standard marching rate for infantry is 120 paces per minute, which is roughly 100m per minute. In a skirmish game, which represents perhaps a total of 10 minutes action, the units would normally run or move at the double, so we could consider that 200m per minute is a reasonable average rate of movement on a road or hard track. If the time scale is *1 minute per turn*, and the map scale is *33m per square*, we arrive at a movement rate of six squares per turn. This basic rate is reduced to four for wounded men, or

reduced to other amounts for moving over rough ground. It is reduced even further for difficult terrain such as jungle or marsh. These were the calculations used when deciding the scales used for JUNGLE ATTACK.

Another point to consider when deciding on scales and on the overall size of the map is the range of weapons used by the troops. For example, the effective range of a modern rifle is approximately 300m. On the scale 1 square = 33m the rifle has a range of nine squares. On the same scale a grenade can be thrown only two squares. These ranges need to be taken into account when planning the size of the map and the size and location of features shown on the map.

Cartographer

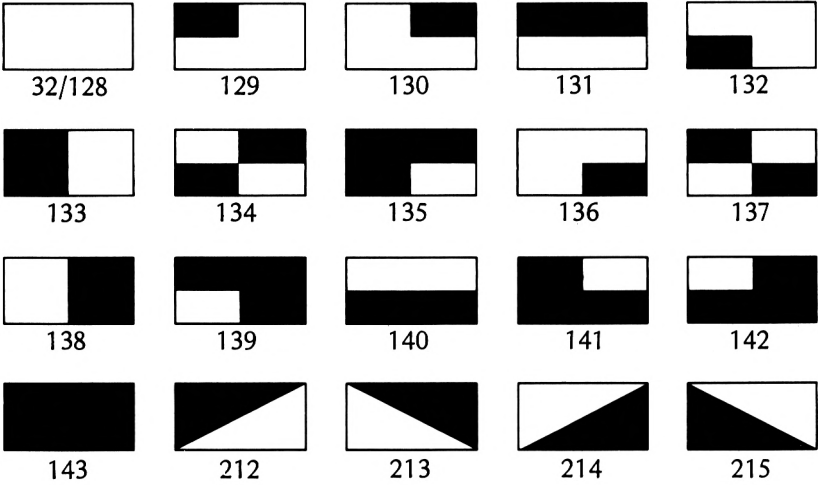
This is a utility program for designing terrain maps and creating a data file of the map. It can also be used for editing details of an existing map data file. Key in the program from the listing given in the next section. You will also need to key in and save the machine code file MCODE as explained in the last section of this chapter. Save this on the same disk as CARTOGRAPHER or immediately after it on tape.

Before using the program, decide the scale and how many columns and rows the map is to have. Map designs are provided for all the games in this book (e.g. Figure 2.1). Next decide the colours to be used for each type of terrain. CARTOGRAPHER and the games programs all use the default colours of Mode 0, except that paper/pen 14 is redefined as white (Table 1). In practice, white (as opposed to bright white) is more like a grey. Yellow appears on our screens as a dark olive green, which makes a suitable colour

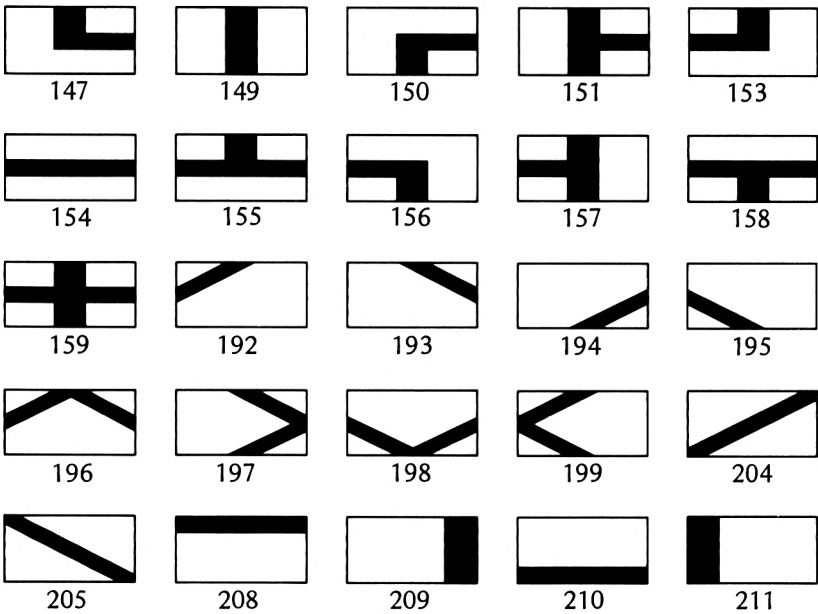
Table 1 Paper and pen colours

<i>Paper/Pen number</i>	<i>Colour</i>
0	Blue
1	Bright yellow
2	Bright cyan
3	Bright red
4	Bright white
5	Black
6	Bright blue
7	Bright magenta
8	Cyan
9	Yellow (= Olive green)
10	Pastel blue
11	Pink
12	Bright green
13	Pastel green
14	White (= Grey)
15	Flashing Pink/Blue

Areas of terrain, buildings etc.



Paths, roads, trenches, earthworks, contours, etc.



Miscellaneous



Fig 3.1 Useful graphics symbols, with their ASCII codes.

for woodland areas. The colours we have used for each feature on the maps in the games are given in each game description. The map is built up from the characters of the Amstrad graphics character set. The most useful of these for mapping are shown in Figure 3.1. The symbols are drawn as they appear when displayed in Mode 0. The map display routines also allow you to use the graphics characters associated with ASCII Control Codes 0 to 31. Some of the more useful ones are illustrated in Figure 3.2. Codes 1–3 and 23

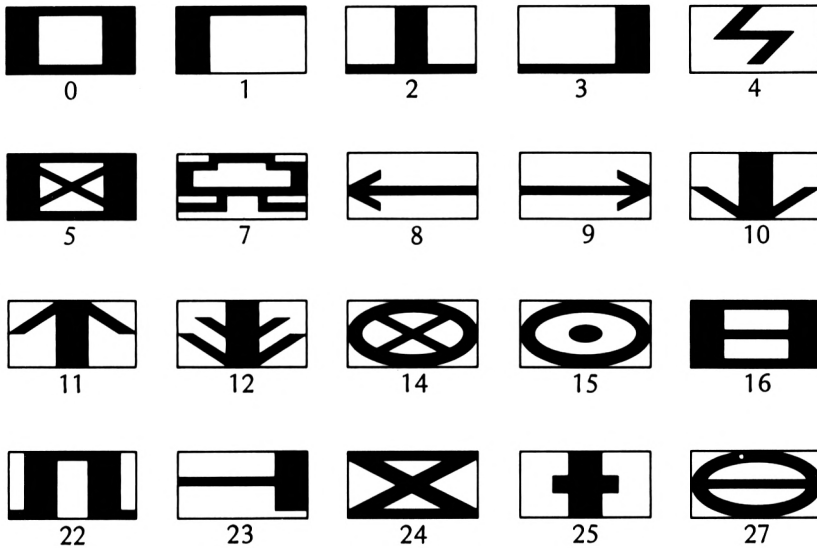


Fig 3.2 Some useful Control Code characters available for mapping.

could represent fences or walls, Code 7 could be a tree, Code 12 is a good one for scrubby vegetation, while many of the others could be used for military installations which are not moved during the game, such as a headquarters' building, a redoubt, or a military hospital (Code 25). ASCII codes 234 to 255 will not normally be used in mapping as these ASCII numbers are reserved for definitions of unit symbols.

The map is displayed in Mode 0, which has 20 characters across the width of the screen. Each square of the map is represented by two characters, one for the top half of the square and one for the bottom half (Figure 3.3). When designing a square remember that, when a unit symbol is on a square, it completely occupies the top half. Thus, if a square is to contain a feature such as a small building, it is better if at least some part of the feature appears in the bottom half of the square. Otherwise, the unit obliterates the feature entirely, causing confusion when reading the map.

When mapping roads or tracks, design the squares so that any unit symbol displayed on the square appears to be *on* the road or track, rather than *beside* it (Figure 3.4). River squares can be designed so that, as far as possible, units appear to be on the bank of the river rather than in it!

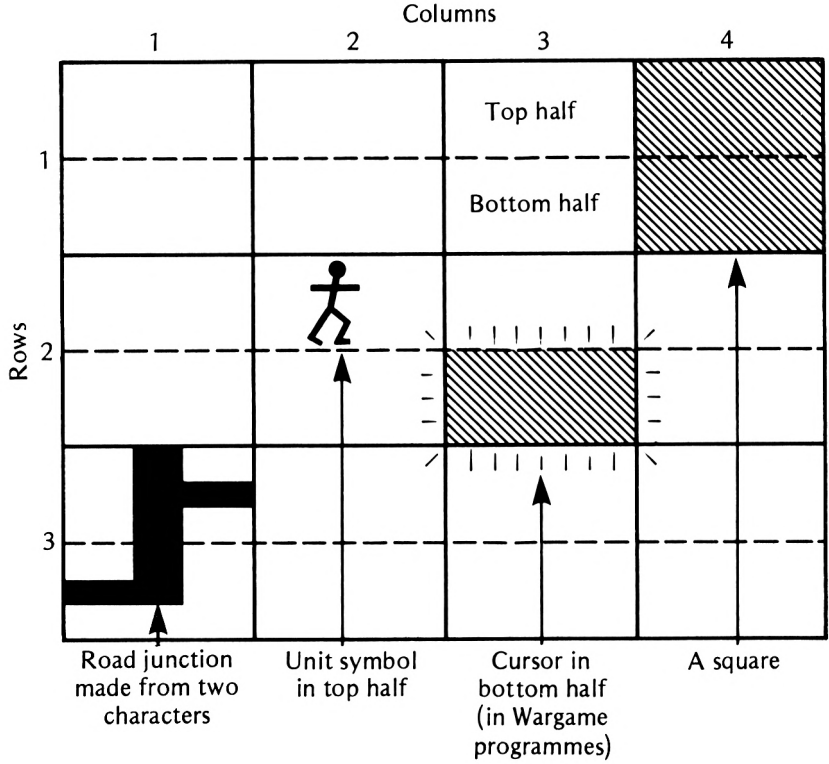


Fig 3.3 The map display system.

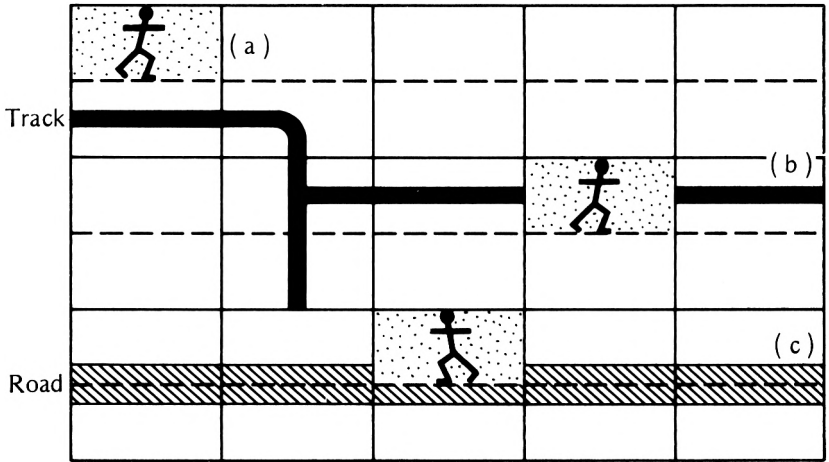


Fig 3.4 Roads and tracks. In (a) the unit appears to be off the road, even though it is actually on the road square. (b) and (c) give a better representation.

Using cartographer

When the program is run the screen clears and is divided into four sections or windows. The main window (blue) is where the section of the map is to be created. A message appears in this window asking "Existing map file?". Type 'Y' if you have already created a map file and wish to amend or correct it. You are asked the file name and the file is loaded from tape or disk. The top left (north-west) section of the map is displayed, ready for editing.

If you want to start a new map file, key 'N' in response to the question above. You are asked to key in the number of columns and rows it is to have, and also the main paper colour. The main paper colour is that which is to cover the greater part of the area of the map. All squares are defined in that colour to start with. This can save you much time, as you will not need to key in further details for these squares individually. The numbers for the colours are in Table 1. When you have keyed in the number, the main window changes to the paper colour you have selected.

Cursor control The cursor is the flashing patch at the top left corner of the screen. It is moved by pressing the cursor control keys (the 'arrow' keys). The first time you use the program, experiment by pressing these keys in turn. As you do so, note the changing numbers displayed in the green window at the bottom right of the screen. These indicate the map column and map row in which the cursor is positioned. The row number changes, once for each two steps of the cursor. This is because in CARTOGRAPHER (but not in the games) the cursor has *two* positions in each map square, top half and bottom half.

If you move the cursor downward when it is at the bottom of the map section, it reappears in the green window at bottom right. Do this when you want to change the display to show another section of the map. For example, to display a section to the south of the present section, position the cursor on the 'S' and press the space bar. The section to the south is then displayed. This new section overlaps the previous section by six rows. Sections to the east or west can be displayed similarly, but here the overlap is 10 columns. At this stage, *do not position the cursor over the 'F'* and press the space bar. Do this only when you have finished mapping and wish to save the map file. Return the cursor to the main window by moving it upward.

Placing a character Figure 3.5 shows how the map of Figure 2.1 may be created using the Amstrad's graphic characters. Move the cursor to the square on which you wish a character to be displayed. Move it up and down one or two steps checking the row number each time, to ensure that it is in the correct half (top or bottom) of the square. Press the space bar. The cursor is replaced by a diagonal cross symbol. The central lower window (grey) displays a series of messages as follows:

Copy? This is for use when repeatedly placing the same character in several squares, as explained later. To select a new character instead of copying one, press the space bar again.

Character? Key in the number of the character required (see Figures 3.1 and 3.2, or the User Instructions). The number must be in the range 0–233. Numbers may be entered on the main keyboard or the key-pad. As soon as

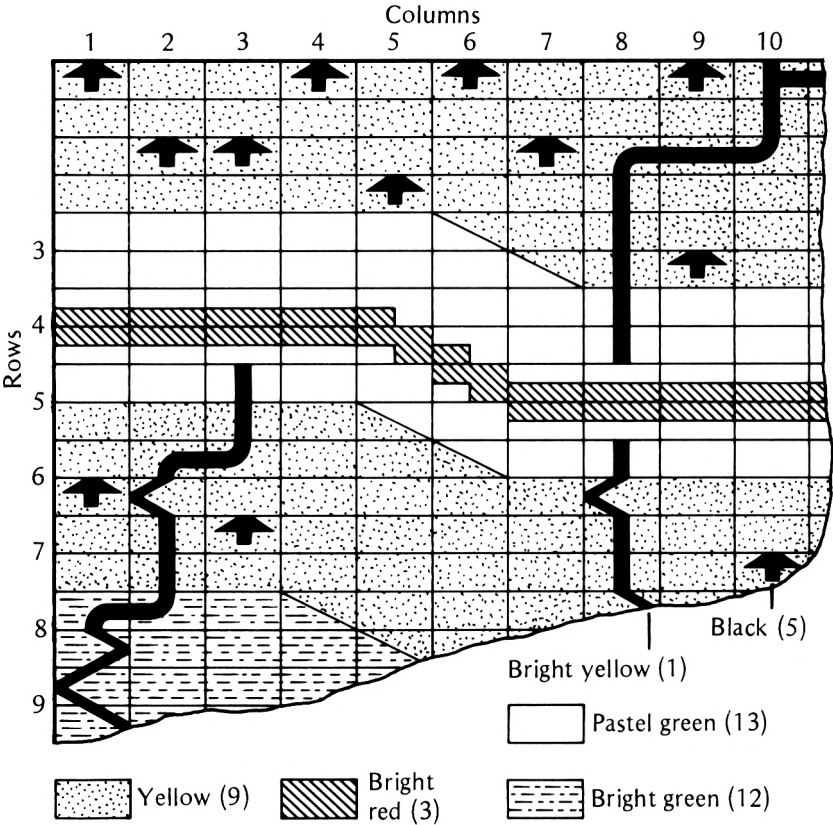


Fig 3.5 How the map of Fig 2.1 may be designed by using the Amstrad's firmware character set, in Mode 0. Numbers in brackets refer to Paper and Pen numbers.

you press ENTER, the chosen character appears in the bright green window at bottom left. It is displayed in black (pen) on white (paper).

Paper? Enter a paper number from Table 1. The paper of the character in the bottom left window changes to the colour you have chosen.

Pen? Enter a pen number from Table 1. The program does not allow pen and paper numbers to be the same. If you key the same number for pen and paper, you are asked to select both colours again. When you press ENTER, the pen colour of the character at bottom left changes to the selected colour. The character also appears on the map at the cursor location. You are not able to see it clearly at first as the cursor is flashing on the same area. Move the cursor away to see the character.

More characters Now that you have defined a character and its colours, you can copy this on to any other squares of the map. Move the cursor to the required position and press the space bar. The 'Copy?' message appears. Press the COPY key to cause the character to be copied at the cursor location. This feature allows you to cover large areas of the map speedily with the same character. When you want to define another

character, proceed as described in the previous paragraph. Remember that alphabetic characters may also be used on the map, especially for naming towns and other features.

Editing Editing is simple and is particularly useful if the map has mistakes, if you wish to alter a feature of it, or if you wish to try out the effects of different characters or combinations of colour. Any character may be overwritten by a different one. Place the cursor on it and copy another character to that position, or define a new character and its colours as described above.

Other map sections Change the section of map displayed by moving the cursor on to 'NS EW' in the bottom right corner and press the space bar.

Finishing When you are satisfied that the whole map is correct, move the cursor to the 'F'. The screen clears. You are asked to key in the file name under which the map data is to be saved. With disks, the file is saved immediately. With tapes, press RECORD and PLAY, then any key. After saving you are given the opportunity to take as many back-up copies of the file as you require. Change the disk or tape before answering 'Y' to 'Backup?'. When you answer 'N', the program finishes.

Listing

```

10 REM *** CARTOGRAPHER ***
20 MEMORY 28999:MODE 0:INK 14,13:BORDER 13
30 WINDOW #0,1,20,1,22:w=0
40 WINDOW #1,1,3,23,25:PAPER #1,13
50 WINDOW #2,4,15,23,25:PAPER #2,8:PEN #2,5
60 WINDOW #4,16,20,23,25:PAPER #4,9:PEN #4,3
70 CLS:CLS #1:CLS #2:CLS #4
80 LOAD "MCODE"
90 CLS:PRINT"Existing map file? (Y/N)":GOSUB 940
100 IF a$="N" THEN 140
110 PRINT:INPUT "File name";F$
120 LOAD f$
130 xs%=(PEEK(30999)-20)*3:GOTO 240
140 CLS:m$="No. of columns":l=20:u=40:GOSUB 910
150 ncol=number
160 m$="No. of rows":l=11:u=INT(1000/ncol):GOSUB 910
170 nrow=number
180 m$="Main map paper no.":l=0:u=15:GOSUB 910
190 pa=number:pe=number:char=32
200 CLS:POKE 30998,nrow:POKE 30999,ncol
210 POKE 31000,number:POKE 31001,number:POKE 31002,32

220 FOR j=0 TO 2:CALL 29100,31000+j,31003+j,nrow*ncol
230 xs%=(ncol-20)*3
240 CALL 29003,31000,xs%
250 PRINT #4,"NS EW":LOCATE #4,3,3:PRINT #4,"F"
260 nrow=PEEK(30998):ncol=PEEK(30999):xc=1:xn=1:yc=
1:yn=1:top=1:lft=1:lwr=nrow-10:rgt=ncol-19
270 GOSUB 980:PAPER 12:IF mo<>0 THEN GOSUB 970
280 IF mo=32 THEN PRINT CHR$(7):LOCATE xc,yc:PRINT CH
R$(203);:GOTO 700
290 LOCATE xc,yc:CALL &BBBA
300 IF mo=242 AND xc>1 THEN xn=xc-1
310 IF mo=243 AND xc<20 THEN xn=xc+1
320 IF mo=240 AND yc>1 THEN yn=yc-1
330 IF mo=241 AND yc<22 THEN yn=yc+1

```

```

340 LOCATE xc,yc:CALL &BBBD:IF mo=241 AND yc=22 THEN
410
350 IF xc=xn AND yc=yn THEN 270
360 LOCATE #4,1,2:PRINT #4,SPACE$(5);
370 PAPER 0:INK 0,1:PEN 1:INK 1,24
380 xm=1ft+xn-1:IF xm<10 THEN LOCATE #4,2,2:PRINT #4,
MID$(STR$(xm),2); ELSE LOCATE #4,1,2:PRINT #4,MID$(ST
R$(xm),2);
390 ym=top+INT((yn+1)/2)-1:IF ym<10 THEN LOCATE #4,5,
2:PRINT #4,MID$(STR$(ym),2);ELSE LOCATE #4,4,2:PRINT
#4,MID$(STR$(ym),2);
400 xc=xn:yc=yn:GOTO 270
410 xc4=3:yc4=1:xn4=3:yn4=1
420 GOSUB 980:PAPER 5:IF mo<>0 THEN GOSUB 970
430 LOCATE #4,xc4,yc4:PRINT #4,CHR$(233)
440 IF mo=242 AND xc4>1 THEN xn4=xc4-1:IF yc4=3 THEN
xn4=3
450 IF mo=243 AND xc4<5 THEN xn4=xc4+1:IF yc4=3 THEN
xn4=3
460 IF mo=240 AND yc4=3 THEN yn4=1
470 IF mo=241 AND yc4=1 THEN yn4=3:xn4=3
480 LOCATE #4,xc4,yc4:IF yc4=1 THEN ON xc4 GOTO 500,5
10,520,530,540
490 PRINT #4,"F";:GOTO 550
500 PRINT #4,"N";:GOTO 550
510 PRINT #4,"S";:GOTO 550
520 PRINT #4," ";:GOTO 550
530 PRINT #4,"E";:GOTO 550
540 PRINT #4,"W";:GOTO 550
550 IF mo=240 AND yc4=1 THEN 400
560 IF mo=32 THEN PRINT CHR$(7):GOTO 580
570 xc4=xn4:yc4=yn4:GOTO 420
580 IF yc4=3 THEN 850
590 ON xc4 GOTO 600,620,640,650,670
600 top=top-6:IF top<1 THEN top=1
610 GOTO 680
620 top=top+6:IF top>1wr THEN top=1wr
630 GOTO 680
640 GOTO 430
650 1ft=1ft+10:IF 1ft>rgt THEN 1ft=rgt
660 GOTO 680
670 1ft=1ft-10:IF 1ft<1 THEN 1ft=1
680 CALL 29003,30997+(top-1)*ncols*6+1ft*3,xs%
690 GOTO 270
700 CLS #2:PRINT #2,"Copy?"
710 a$=INKEY$:IF a$="" THEN 710
720 IF ASC(A$)=224 THEN 820
730 pa=4:pe=5: w=2:m$="Character":l=0:u=233:GOSUB 910

740 char=number
750 PAPER #1,pa:PEN #1,pe:LOCATE #1,2,2:PRINT #1,CHR$(
1);CHR$(char);
760 m$="Paper":l=0:u=15:GOSUB 910
770 pa=number:IF pa=5 THEN pe=4
780 PAPER #1,pa:PEN #1,pe:LOCATE #1,2,2:PRINT #1,CHR$(
1);CHR$(char);
790 m$="Pen":l=0:u=15:GOSUB 910
800 pe=number:IF pe=pa THEN pa=4:pe=5:GOTO 750
810 PAPER #1,pa:PEN #1,pe:LOCATE #1,2,2:PRINT #1,CHR$(
1);CHR$(char);
820 PAPER #0,pa:PEN #0,pe:LOCATE #0,xc,yc:PRINT #0,CH
R$(1);CHR$(char);
830 se=30994+(top-1)*ncols*6+1ft*3+((yc-1)*ncols+xc)*
3:POKE se,pa:POKE se+1,pe:POKE se+2,char
840 CLS #2:GOTO 270
850 PAPER 0:PEN 1:MODE 1:LOCATE 2,2:INPUT"File name";
F$
860 LOCATE 2,4:SAVE F$,B,30998,2+nrows*ncols*6

```

```

870 CLS:LOCATE 2,2:PRINT"Backup? (Y/N)":GOSUB 940
880 IF a$="Y" THEN 850
890 CLS:LOCATE 2,2:PRINT"Cartographer finished"
900 LOCATE 1,23:END
910 number=-1000:WHILE number<1 OR number>u:REM SELEC
TNUMBER ***
920 CLS #w:PRINT #w, m$:PRINT #w,"(";1;" to ";u;");
930 INPUT#w,#;number$:number=VAL(number$):WEND:RETURN

940 a$=UPPER$(INKEY$):REM Y/N ***
950 IF a$<>"Y" AND a$<>"N" THEN 940
960 RETURN
970 SOUND 1,60,1:RETURN:REM PIP ***
980 j$=INKEY$:IF j$="" THEN mo=0:RETURN:REM MOVE ***
990 mo=ASC(j$):RETURN

```

Program description

20–70 Protect memory, set up windows and colours.
 80 Load machine code routines.
 90–130 Loading existing map file, if any.
 140–180 Inputting columns, rows, and main paper colour.
 190–220 Filling map memory with 'spaces' in main paper colour, using m/c BLOCK.
 230 Set xs%240 Display map, using m/c SMAP.
 250 Display NSEW and F.
 260 Set screen parameters.
 270–400 Cursor movement on map section; lines 380–390 calculate and display column and row numbers.
 410–570 Cursor movement in bottom right window.
 580–690 Recalculate screen parameters and display new map section.
 700–720 Selecting Copy option.
 730–790 Inputting character details.
 810–840 Displaying character at bottom right and on map.
 850–900 Saving and backup.
 910–930 Subroutine for accepting number within given range.
 940–960 Subroutine for Y/N input.
 970 Sound subroutine for cursor movement.
 980–990 Subroutine for cursor control key input.

MCODE

CARTOGRAPHER and the games programs require a few machine-code routines, which are saved as a single binary file called MCODE. Although only two of these routines are required by CARTOGRAPHER, it saves time and effort to key in all the routines now, ready for use later in games. Use one of the two methods below to prepare the MCODE file:

Method 1 Key in and save the cipher program MCODEC listed below. When this is run, it creates the MCODE file on tape or disk. You now have

the file MCODE on your tape and can copy it on to games programs tapes as described below.

It is best to save the MCODEC program even though it is required only once, when creating MCODE. Should the machine code programs subsequently fail to work properly, reload MCODEC and check it very carefully, as the fault is probably due to typing errors. When you have corrected and re-saved MCODEC, run it again to create a new corrected version of MCODE.

```

10 REM *** MCODEC ***
20 MEMORY 28999
30 FOR j=29000 TO 29392
40 READ x$:x$="&" + x$:POKE j,VAL(x$)
50 NEXT
60 STOP
1000 DATA 00,00,00,FD,21,48,71,DD,7E,00,FD,77,02,DD,6E
1010 DATA 02,DD,66,03,E5,DD,E1,3E,00,CD,B4,BB,CD,6C,BB
1020 DATA 3E,16,FD,77,00,3E,14,FD,77,01,DD,7E,00,CD,96
1030 DATA BB,DD,7E,01,CD,90,BB,DD,7E,02,CD,5D,BB,DD,23
1040 DATA DD,23,DD,23,FD,35,01,C2,70,71,FD,35,00,C8,FD
1050 DATA 4E,02,06,00,DD,09,C3,6B,71,2A,8E,71,FD,21,90
1060 DATA 71,3E,16,FD,77,00,3E,14,FD,77,DD,7E,00,32,CC
1070 DATA 71,DD,7E,01,32,CD,71,DD,4E,06,DD,46,07,DD,5E
1080 DATA 08,DD,56,09,DD,6E,0A,DD,66,0B,7E,CD,EC,71,12
1090 DATA A7,DD,7E,04,85,6F,30,01,24,A7,DD,7E,02,83,5F
1100 DATA 30,01,14,0B,3E,00,B9,C2,CA,71,BB,C2,CA,71,C9
1110 DATA E6,BF,C9,00,00,00,00,00,00,00,00,00,00,00
1120 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00
1130 DATA 00,00,00,00,00,DD,5E,00,16,00,DD,6E,02,DD,66
1140 DATA 03,CD,AB,BB,C9,00,00,00,00,00,00,00,00,00
1150 DATA 00,00,00,00,00,00,00,00,00,00,2A,26,72,22,28,72
1160 DATA 3A,2C,72,32,2D,72,DD,21,2E,72,00,3A,2A,72,6F
1170 DATA ED,4B,24,72,ED,5B,26,72,0A,A7,CA,94,72,32,2E
1180 DATA 72,1A,A7,CA,94,72,32,30,72,DD,96,00,F2,6C,72
1190 DATA 3A,2E,72,A7,DD,96,02,BD,F2,94,72,03,13,0A,A7
1200 DATA CA,94,72,32,2F,72,1A,A7,CA,94,72,32,31,72,DD
1210 DATA 96,01,F2,8F,72,3A,2F,72,A7,DD,96,03,BD,F2,94
1220 DATA 72,C9,06,00,0E,09,2A,26,72,A7,ED,42,22,26,72
1230 DATA 3A,2C,72,3D,32,2C,72,C2,3E,72,3A,2D,72,32,2C
1240 DATA 72,ED,5B,28,72,ED,53,26,72,3A,2B,72,3D,32,2B
1250 DATA 72,CB,06,00,0E,09,2A,24,72,A7,ED,42,22,24,72
1260 DATA C3,3E,72

```

Method 2 If you have an assembler program, key in and save the code, using the hexadecimal values in the data statements above. Code should begin at &7148 (29000), and occupies 393 bytes.

Copying MCODE

- 1 Press CTRL, SHIFT and ESCAPE, in that order to clear the computer.
- 2 Place tape or disk with MCODE in the recorder or disk drive.
- 3 Type MEMORY 28999 and press ENTER.
- 4 Type LOAD "MCODE" and press ENTER. The file is loaded. (On tape, press PLAY and any key.)

5 Place the new tape or disk in the recorder or drive. On tape, MCODE is usually saved directly after the game program, so run the tape on to just beyond the end of the games program.

6 Type SAVE "MCODE",B,29000,393 and press ENTER. The file is saved. (With tape, press RECORD and PLAY and any key.)

The MCODE routines

The notes below are intended for those who wish to produce their own games and incorporate MCODE routines. The four routines are SMAP, BLOCK, SETMAT and PROX.

SMAP Displays a section of the terrain map 20 columns wide and 11 rows deep in the top 22 screen lines. Code is from 29000 to 29083, entry point is 29003. See Chapter 5 for details of how map data is stored.

Calling: The micro must already be in Mode 0. Calling statement is:

CALL 29003,start%,xs%

Where start% is the address of the first byte defining the square at the top left corner of the map section to be displayed, and xs% is the number of columns of the map not being displayed. Calculate start% as in line 680 of CARTOGRAPHER, and xs% as in line 230.

BLOCK A general-purpose routine for transferring and processing blocks of memory. It is slower than a straightforward block transfer routine, which is of no consequence in wargame programs, but has several features that give it versatility. In essentials, it moves a block of memory of any size (b% bytes), one byte at a time, from one location in memory (in the 'from' block, base address f%) to another (in the 'to' block, base address t%). Transfer begins with the lowest address within each block. Instead of operating on successive bytes in the block, the routine can step through either or both blocks, transferring, for example, every second byte, or every tenth byte, to stepped addresses in the 'to' block. The step parameters are sf% in the 'from' block and st% in the 'to' block.

The routine is able to process each byte during transfer between blocks. The byte is loaded from the 'from' block into the accumulator. A subroutine is called to perform a given operation on the byte before it is loaded from the accumulator into the 'to' block. In the simplest case, the subroutine is a RET, and the data is transferred without alteration. The subroutine to be used is determined by setting the vector variable v% to the entry address of the subroutine. For transfer of data unchanged, set v% to 29164. The routine is stored at 29100 to 29164, entry point 29100. Memory from 29165 to 29199 is reserved for the subroutines. Of this area, addresses 29165–29167 are used for the RALLY subroutine (see below).

Calling: CALL 29100,f%,t%,b%,sf%,st%,v%. Examples of use of this routine are:

(a) Simple transfer of a block of 200 bytes starting at 45000 to a new location starting at 37000; CALL 29100,45000,37000,200,1,1,29164.

(b) Filling a block of b% bytes with a given value. Poke the start address (start%) of the block with the value. Then CALL

29100,start%,start%+1,b%,0,1,29164. The step value of the 'from' block is zero, so the same value is placed in all addresses of the 'to' block.

See also lines 210–220 of CARTOGRAPHER which fill a block with a repeated sequence of three values: paper, pen, ASCII code.

(c) RALLY subroutine, takes the status byte of each unit of an army and ANDs it with 191, to reset the 'panic' bit to zero. CALL 29100,aa(na)+15,aa(na)+15,units(na),9,9,29165. Here the 'from' and 'to' blocks are the same block. Data in every 9th address is read, ANDed and replaced in the same address.

SETMAT Sets the start of the character matrix (block of memory in which user-defined graphics characters are to be stored). This routine is used instead of the BASIC keyword, SYMBOL AFTER, which will not operate correctly in the games programs since memory boundaries are different from usual. Stored at 29200 to 29214. Entry point 29200.

Calling: Suppose that the matrix is to be located immediately below address 29000, as in the games programs. If *asco* is the ASCII code of the character with the lowest ASCII code, the start address for the character matrix is:

$$\text{start} = 29000 - (256 - \text{asco}) * 8$$

Begin the program with the statement MEMORY *asco*–1. To set the matrix, CALL 29200,start, *asco*.

See subroutine SYMBOLS of the ATTACK program.

PROX Searches the data of two armies to find pairs of units, one unit from each army, that are in proximity, i.e. which are occupying the same square, or are within a certain distance of each other. This routine is used in looking for instances of close combat, for example, or for finding which pairs of opposing units may be hidden from each other because the distance between them exceeds a given limit. Stored at 29220 to 29392. Entry point 29234; re-entry point 29332.

Calling: We refer to the two armies as Army 1 and Army 2. Before calling perform the following POKEs:

29220/1 The address of the row byte of the last unit of Army 1 (low byte/high byte, as usual). This address is $aa(na)+3+9*units(na)$.

29222/3 Ditto for Army 2.

29226 The limit, the distance at or beyond which the units are not regarded as being in proximity. The limit is expressed in squares. To find units on the same square, the limit is 1. Do not set the limit to zero.

29227 The number of units in Army 1 ($units(1)$)

29228 Ditto for Army 2.

After poking the above, the first call to the routine is CALL 29234. Then PEEK 29227 and 29228. If 29227 equals zero, no pairs have been found in proximity. Otherwise, 29227 gives the number of the unit of Army 1 and 29228 gives the number of the unit of Army 2. If it is required to know the row and column on which these units are situated, these may be obtained by PEEKing:

29230 Row of Army 1 unit

29231 Column of Army 1 unit

29232 Row of Army 2 unit

29233 Column of Army 2 unit

If further pairs of units are to be found, re-call (without repeating the POKES) by CALL 29332. This may be repeated until the routine returns with PEEK 29227 equal to zero, indicating that no further pairs have been found.

4 Resolving combat

The outcome of combat between two units or forces depends on numerous factors. Most of these factors can be expressed in numerical or logical terms and board wargamers, is to refer to a combat resolution table. Although it is possible to program the computer to be used like a combat resolution table, and board wargamers is to refer to a combat resolution table. Although it is possible to program the computer to be used like a combat resolution table, this is not making the best use of the computer. Before pocket calculators and computers were available, people used tables of square roots, logarithms, antilogarithms, sines and cosines to assist them in resolving mathematical problems. As soon as calculators were developed with such functions obtainable at the touch of a button, mathematical tables dropped out of use. In the same way, we should make combat resolution available to wargamers at the touch of a key or, preferably, as a completely automatic routine forming part of the wargame program. In this chapter we examine various ways of doing this.

We are concerned with three main types of combat:

- 1 Combat between individuals, using missile weapons, such as slings, bows, rifles, mortars, and artillery.
- 2 Combat between groups, (squads, companies, regiments, armies) using a variety of missile weapons and non-missile weapons, such as swords, pikes and daggers.
- 3 Combat using other weapons, such as mines and demolition charges. In dealing with third category of combat, emphasis is on programming logic. Conditional statements (e.g. IF . . . THEN . . . ELSE) decide when the weapon is to explode and what its effects are to be. There are few general principles involved and routines to simulate these weapons need to be specially tailored to the scenario of the game in which they are used. We shall not discuss this type further. The other two types of combat are amenable to treatment as all-purpose algorithms, as described below.

Missile weapons

This section considers the possible effect of a single missile being projected at its target. The discussion centres around rifle fire. Similar considerations

apply to all missile weapons from blow-pipes to hydrogen bombs but not to guided missiles.

The first step is to decide whether or not the missile actually hits the target. If a target is fired at repeatedly with a rifle, the shots land in a cluster around the point of aim (assuming there is no systematic error, such as that due to an inaccurate gunsight). The hits are not evenly concentrated around that point. As Figure 4.1 shows, the concentration of hits is greatest close to

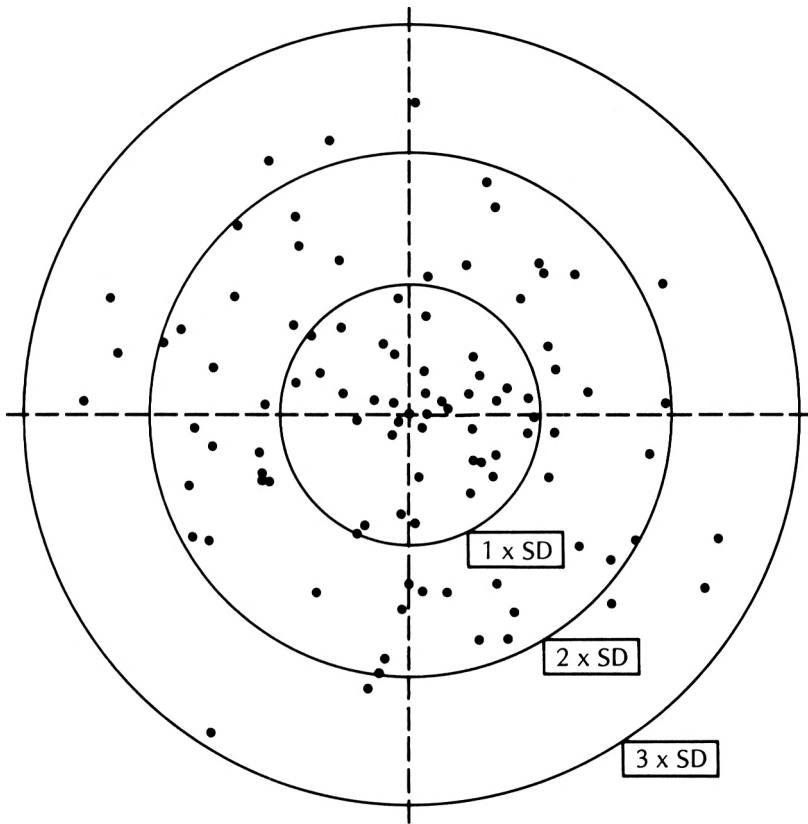


Fig 4.1 The result of target-practice, using the RANORM function for 100 shots.

the centre of aim and falls off at increasing distances from it. The amount of spread of hits determines what proportion of these fall on a target of given size. Statisticians have devised a statistic which they use for specifying the amount of spread. They call it *standard deviation*. We need not be concerned about the exact meaning of this term or how to calculate it, for later we shall be simply guessing a suitable value for it and then testing to see whether it produces a reasonable result. But the concept of standard deviation underlies the discussion which is to follow, so we are obliged to mention it. For brevity, we will refer to it as SD. The variable name *sd* is used to represent it in the programs. As an illustration of its meaning, we may say

that, if we draw a circle around the 45% of the hits nearest the centre of the cluster, and measure the radius of this circle, this figure gives the value of SD. If a circle of radius twice SD is drawn around the same cluster, it includes 90% of hits. Note that although the second circle has an area four times that of the first, it does not encircle four times as many shots – only about twice as many. The shots are concentrated inside the inner circle, close to the centre of aim. A circle of radius three times the SD includes over 99% of shots. In other words, it is rare for a shot to land three times the SD from the target. No one could miss so badly!

The value of the SD depends on a number of factors, primarily the type of weapon used. A 17th Century matchlock musket had a large SD (wide spread of hits at a given range), while a modern rifle has a small SD. A modern revolver has a relatively large SD, mainly because of its short barrel. The SD also varies according to the firer. When a novice is firing, the SD is much greater than when aim is taken by a crack shot. The wavering aim of a seriously wounded man increases SD further. Given the SD associated with the type of weapon and skill of aiming it, and given the size and the range of the target, our task is to calculate the chance of hitting the target. Figure 4.2 illustrates two extremes. At close range, all shots within three SDs of the aiming point strike the target. There is a 100% chance of a hit. At a greater range only those within one SD hit the target. The chance of a hit is about 45%. Our next step is to find out what will happen at any given distance with a target of any given size.

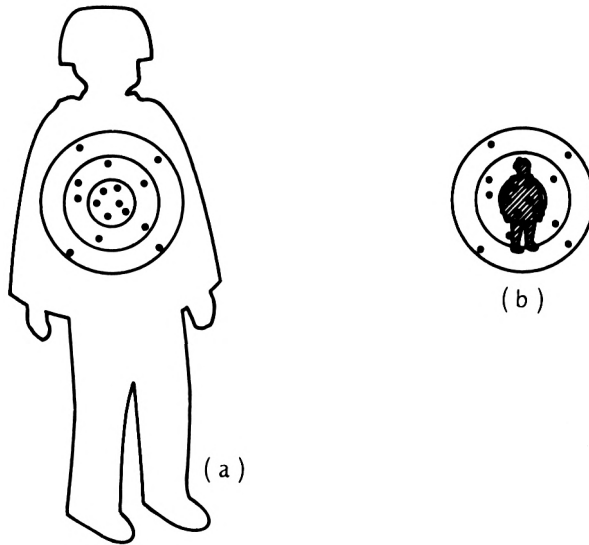


Fig 4.2 Effect of range on chance of hit.

From the above it is clear that, when programming combat resolution, we need to arrive at a value for SD which gives the same results as would be obtained by an average, trained person firing the real weapon at a real

target under average conditions. Here we come against a difficulty. The compilers of combat resolution tables are in the same predicament. Very little accurate information is available on the effectiveness of weapons. Moreover, performance under battle conditions may differ markedly from that recorded under test conditions. Information on the performance of ancient weapons is scanty while the results of testing modern weapons are heavily guarded secrets. The best we can attempt to do is to try to arrive at results which accord with experience, and which 'seem right' in the context of a particular wargame. That is not to say that we are free to resolve combat in any way we please. The algorithm for a weapon must 'behave' like the actual weapon. It must give results which are realistic, as far as the players are able to judge.

In the discussion above, the chance of hitting the target was expressed as a percentage, on the basis of the target being fired at hundreds of times. What we really want to decide in a wargame is whether a particular single shot will hit the target or miss it. In real life the only way to find that out is to take aim and fire. On the computer, a special algorithm is used to obtain the same effect. First we try to calculate how far from the point of aim (normally the centre of the target) the shot will fall. This is roughly equivalent to asking: "Of the 100 typical shots in Figure 4.1, which one is the one we are firing *now*". The positions of the shots in Figure 4.1 were determined by using a special kind of random number to fix the two coordinates of each shot. The properties of the random number used are such that its values cluster around zero in a similar way to that in which rifle shots cluster around the point of aim on a real target. Values of the random number are equally likely to be positive or negative. There are more values closer to zero than there are further from zero. The special type of random number we require is obtained by using the *random normal function*, *RANORM*. Wargamers often generate a similar type of random number using two "normal" dice. A simple method of generating such a number was described by Arthur G. Hansen in *Byte*, October 1985. He uses the ordinary RND function to obtain four random numbers (in the range of 0 to 1) in succession, and takes the average of these. The average is very unlikely to be close to one, since this would mean obtaining four random numbers close to one, in succession, which is unlikely. For similar reasons it is unlikely that the average will lie close to zero. Averages obtained in this way will be clustered around 0.5. If we subtract 0.5 from each of these averages we obtain values clustering around zero. Theory tells us that the SD of a set of values obtained in this way is 0.144. Dividing every value by 0.144 gives us a set of random numbers clustered around zero with SD of one. A routine for generating such random numbers is ideal for use in deciding whether a shot hits its target or not.

In Figure 4.1 we used two such numbers for determining the position of each hit, one for the horizontal coordinate, one for the vertical coordinate. It was assumed that the SD in the horizontal direction equals that in the vertical direction, i.e. that the weapon produces a cluster that is circular in outline, not elliptical.

To simulate the way in which shots are clustered on a target we need to be able to obtain random numbers which have the required SD. We first

have to decide what SD these numbers should have. As explained earlier, little information is available to help us. The best we can do is to decide a value using experience as a guide, and if necessary amend the value after testing the program. If numbers obtained by using RANORM are multiplied by the value we have decided upon for SD, we obtain a set which has an SD of our chosen value. If two such numbers are each multiplied by the range, we obtain the horizontal and vertical distances of the shot from the centre of aim. The unit in which these distances are expressed is that in which the range is expressed. If we express range in 'squares' the distances are also in 'squares'.

Let us consider the horizontal distance, which tells us how far to the left or right of target centre the shot falls. For an upright human target we can reasonably stipulate that shot falling within 0.2m to left or right of the centre counts as a hit. Shot falling outside those limits counts as a miss. To compare this limiting distance with values obtained above, we must convert it to the same unit, the 'square'. In JUNGLE ATTACK, for example, the 'square' is equivalent to 33m. On this scale, 0.2m is equivalent to 0.006 squares. If the distance to left or right of the target centre is more than 0.006 squares, the shot is a miss. Scattering in the vertical direction can similarly be taken into account. If the average target height is 1.7m, the shot must fall not more than 0.85m from the centre. In squares, this is 0.026.

The RANORM function is defined on line 3110 of the START subroutine of JUNGLE ATTACK. The combat resolution technique described above is used in the FIRE subroutine, lines 3420 to 3920. First the program sets the values of *sd* according to the weapon fired. This is least for a rifle and greatest for a thrown grenade. Line 3680 multiplies *sd* by a factor which allows for the weapon skill of the firer. High weapon skill reduces *sd* and low weapon skill increases it. The same line also increases *sd* if the firer is wounded. Line 3720 determines the distance of the shot from the target centre; a random number is selected by RANORM and multiplied by *sd* and the range *ra*, to give *pdh*, the probable distance in the horizontal direction. A similar calculation is performed to find *pdv*, the probable distance in the vertical direction. If the absolute value of either of these is greater than a given amount (0.006 for *pdh* and 0.026 for *pdv*), the shot misses.

The routine works realistically at distances up to about half the maximum range. This is the range within which most firing occurs. Moreover, it is within *this* range that the results produced by this algorithm differ most markedly from those produced by other algorithms which assume, incorrectly, the simple 'tapering off' of chance of hits with increasing range. However, our algorithm, as we have described it so far, ignores the effect of maximum range. It allows us to hit the target occasionally, even if it is at infinite distance! Beyond half maximum range it is more acceptable to taper the probability down from its half-range value (*pn*) to zero at maximum range (*maxr*). The subroutine sets values of *pn* and *maxr* for different weapons. Line 3690 adjusts *pn* according to the weapons skill and status of the firer. If the range is greater than half of maximum range, the alternative "tapering algorithm" is used. Line 3710 calculates *p*, the probability of a hit at that distance. This is compared with the random number to decide at random whether the final outcome is a hit or a miss.

If it has been decided that the shot missed the target, there is nothing more to be done. If it has been decided that the target has been hit, we still have to discover what effect this has on the status of the target. From now on, programming is more a matter of logic than computation. The effect of a hit mainly depends on whether the target is protected by cover and, if so, by what kind of cover. In JUNGLE ATTACK there is a table of data giving the cover value of each square of the map. The cover value is 0 (no cover), 1 (light cover, such as trees) or 2 (heavy cover, such as buildings). Line 3730 finds this value, with the additional proviso that the cover value is 2 if the unit is dug in. In the open, the probability of a hit target being killed outright is 0.1 (10%), otherwise the target is wounded (line 3750). Under light cover (lines 3780–3800) the chance of being killed is 0.007 (7%), the chance of being wounded is 0.53 (53%), the chance of it being panicked is 0.2 (20%), otherwise it is unharmed, being protected by the cover. Under heavy cover the figures are 0.005, 0.005, 0.1, with an 80% chance of escaping harm. Having determined the fate of the target, the remainder of the routine is concerned with putting this into effect. Making the necessary alterations to the status data of the target unit is easily done by a few program lines. The resolution of combat can thus be performed entirely by the micro.

It must be stressed that all values used in these routines were first arrived at by a process of combining experience with common sense. Later, when the game was under test, they were adjusted in order to produce a realistic and well-balanced game.

Projectiles

The algorithm described above is intended to model point-blank fire, in which the weapon is aimed horizontally, directly at the target. We have also to consider weapons such as catapults, mortars and artillery in which the missile is projected at a relatively steep upward angle to hit a target which may perhaps be out of sight. Strictly speaking, grenades come into this category, but they have a relatively short range which means that any calculation based on 'squares' is a rather approximate one. It is more convenient and just as precise to program grenades in the same way as rifles and machine-guns.

With mortars and artillery we are primarily concerned with setting the vertical angle (elevation of the weapon) and the horizontal angle (direction of target). Both settings are subject to random variation. These variations can be estimated using the algorithm already described, but with different SDs for each angle. Variations in the horizontal angle produce scattering of the shot to the left or right of the target, as described above for point-blank gunfire. Variations in the vertical elevation of the weapon also produce scattering, usually referred to as *overshoot* and *undershoot*. Since the path of a projectile such as a mortar bomb or shell is a parabola, the effects of variations in elevation are not the same as for point-blank fire. Figure 4.3 shows how equal variations of angle do not produce equal amounts of overshoot or undershoot. If the elevation is greater than 45°, the effect is

reversed. The parabolic path of the missile results in an approximately elliptical area of fall. This is more pronounced the smaller the angle of elevation. To take these effects into account we apply the equations governing the motion of a projectile. We shall ignore factors such as air resistance.

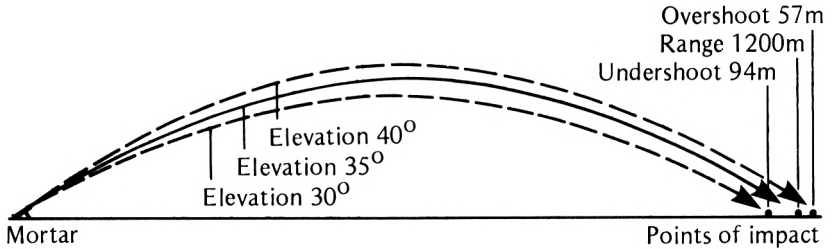


Fig 4.3 Equal variations of angle do not produce equal amounts of undershoot and overshoot.

If the velocity of a shell as it leaves the muzzle of the gun is u , g is the gravitational acceleration (9.81 msec^{-2}), and a is the angle of elevation, the maximum range of the weapon, $r_{\max} u^2/g$. This range is obtained by tilting the weapon upwards at 45° . Having calculated maximum range, we can find r , the range of fall of shot for any other angle of elevation, using the equation, $r = r_{\max} \times \sin 2a$. Before using this equation, we add or subtract any random variation in the angle, as obtained by using the RANORM function. Thus we can calculate exactly where the shot will fall.

In dealing with a mortar bomb or artillery shell we cannot ignore the projectile that misses its intended target. It is certain to land somewhere and may fall on another enemy unit or, worse, on a friendly unit. We must determine exactly where it falls, whether this be on the target or off it. Enemy or friendly units present on the square on which it falls are to be treated just as if they were the intended target.

The MORTAR routine of JUNGLE ATTACK is an example of modelling projectile fire. In line 4100, r_{\max} , the maximum range is 50 squares. sda and sdb are the SDs of the vertical and horizontal angles respectively. These values are selected by trial and error to obtain the desired amount of spread (Figure 4.4).

Line 4100 increases the SDs if the soldier firing the mortar is wounded. The subroutine then calls a further subroutine, PROJECTILE, which performs the essential calculations to determine where the bomb will fall. This calculates range (line 4280) from the differences of x and y coordinates of the location of the mortar and target. Then lines 4290 to 4320 calculate $angb$ the horizontal angle. This is 0 radians (all calculations are in radians for simplicity) for due north, and π radians for due south, measured clockwise. Line 4330 calculates the vertical angle, using an algorithm for the arcsin function (not directly available in Amstrad BASIC). In this, rf is

the actual range of the target divided by the maximum range of the mortar. Line 4340 calculates the random error in the vertical angle, using the

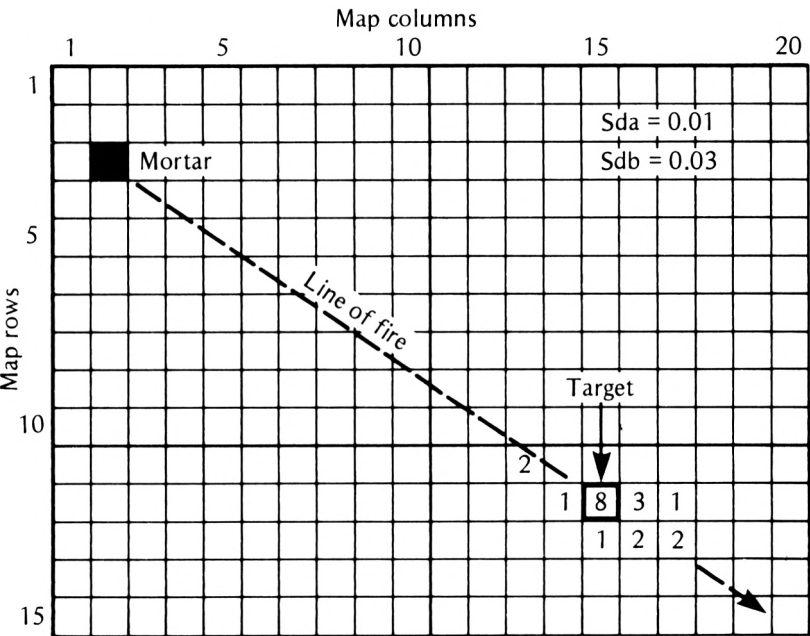


Fig 4.4 Results of twenty consecutive ‘Firings’, using subroutine PROJECTILE. The numbers in the squares indicate the number of bombs falling in that square.

RANORM function, and applies this in calculating the distance from the mortar at which the shot will fall. Line 4350 calculates the random error in the horizontal angle. Finally, line 4360 calculates the coordinates of the square in which the shot falls.

The main MORTAR routine then searches the army data to find if any unit of either army is on this square. If so, it is killed. The routine also looks for units on adjacent squares. These are either killed or wounded, there being a 50% chance of either occurrence. This simulates the likely consequences of being within the burst-circle of a light mortar bomb.

Group combat

In resolving group combat we wish to discover the effect of repeated fire from a number of weapons over a period that may range in length from a few minutes to several hours. In many ways this is easier than trying to calculate the trajectory of a single missile. The first step is to rate each type of unit by its *combat efficiency value* (CEV). This value expresses how many enemy personnel are likely to be killed or otherwise put out of action (seriously wounded, badly panicked), by being fired at by one soldier

during the period of one game turn. The CEV is (like most other parameters of wargaming) based on a combination of published data, wargaming experience and intelligent guesswork! Its value can take many factors into account: the nature of the weapon (which may be a non-missile weapon), or weapons, used by members of the unit, the level of weapons training of the unit, the average range at which the weapon is being used, visibility, seasonal weather conditions, and many other factors.

If the CEV includes factors such as range and visibility, which may change at different stages of the game, we may need to specify a different CEV table for each stage.

In a CEV table, we usually need to specify a CEV for every combat situation that can arise in the game. For example, the CEV of infantry (musketeers and pikemen) in contact with similar infantry is different from that of infantry against cavalry, as Table 2 illustrates. This is the table of CEVs used in NASEBY (Chapter 8). The values in Table 2 are 1000 times the actual CEVs. The tabulated values are stored in bytes of memory, then divided by 1000 before being used. The values relate to a frontal attack, in which the units are facing each other on adjacent squares. The average firing range is just under 100 m. The effective range of the pistols carried by cavalry is less than this, but we assume that they are drilled to move to the front and fire in turn.

The table has a column and row for each type of differently armed and differently trained unit in the game. There is a CEV for each type of unit attacking each type of unit of the enemy. Type 1 (the leaders) are omitted from the table as they do not take part in combat in this game. In the column for Type 2 (infantry with two musketeers to each pikeman) we see that during one game turn we expect each man in the unit to kill or otherwise render ineffective 0.060 enemy infantrymen, 0.084 cavalrymen (larger target, but moving), or 0.030 men defending a baggage train (protected by the wagons). The values take into account the fact that one third of the attacking unit are pikemen, and therefore are incapable of inflicting casualties at that range. Although infantry units of Type 3 have relatively fewer musketeers, their CEVs are slightly higher than those of Type 2. This is because the Parliamentary infantry (Type 2) were newly recruited, while the Royalist infantry (Type 3) were experienced veterans, a difference that was of importance in the battle. Thus the experience of the Royalist musketeers is considered almost to compensate for their smaller numbers.

For the infantry units which are wholly musketeers (Type 4, which includes skirmishers and unmounted dragoons) the CEVs are reduced slightly to take account of the fact that the skirmishers are in open order and do not fire in volleys, while the dragoons are armed with lighter muskets than the musketeers. The column for cavalry (Type 5) shows that their capability of inflicting casualties is lower than that of infantrymen. Their only weapon usable at this range is the pistol, which is considerably less effective than a musket. The mounted dragoons (Type 6) carry a light musket, but need to dismount to fire it. Thus they are ineffective in the combat conditions to which the table relates. The baggage train (Type 7), defended by musketeers has CEVs of average value.

Table 2 Combat efficiency values
For combat at the range of 1 square.

	Attacked unit type						
Attacking unit type	1	2	3	4	5	6	7
1 Leader	0	0	0	0	0	0	0
2 Infantry 2m:1p	0	60	60	60	84	84	30
3 Infantry 1m:1p	0	64	64	64	90	90	32
4 Infantry 1m:0p	0	73	73	73	107	107	34
5 Cavalry	0	43	43	43	53	63	22
6 Dragoon, mounted	0	0	0	0	0	0	0
7 Baggage guard	0	61	61	61	90	90	32

m = proportion of musketeers in unit
p = proportion of pikemen in unit

Table 2 has many of the features of a combat resolution table. Indeed conventional combat resolution tables can offer helpful guidance to the wargame programmer when deciding upon suitable CEVs. Table 2 incorporates all the factors of combat that are not readily expressed in terms of mathematical formulae. Certain random effects have already been taken into account in compiling the table. When we say that an infantryman kills, on average, 0.073 cavalrymen in five minutes, we are taking into account the random effects connected with taking aim and firing, the random chance of a misfire, and several other random factors. We are reasonably sure that although any one infantryman may not hit any opponent during the turn, another of his company may be lucky and hit two. On average, if 300 men fire, the number of opponents hit is 300 times 0.073, a total of 22 men. This is analogous to saying that, if you toss a coin 300 times it always falls with ‘heads’ uppermost on 150 occasions. Although 150 ‘heads’ is the most likely to occur, *on average*, any particular set of 300 throws may well produce a slightly different result. In the same way, we do not expect precisely 22 men to be killed in the turn. The number will be close to 22, but could range from perhaps 16 to 28. Numbers will be clustered around 22, with a relatively small chance of the result being as low as 16 or as high as 28. We need the RANORM function to simulate the clustering effect, as described in the previous section of the chapter. If the SD is set to about 2% of the number of casualties calculated from CEV, it gives a reasonably sized random variation in the result.

Having obtained a CEV based upon the nature of the attacker and the attacked, and having modified this by using RANORM, and possible adjustments to take account of certain features of the game, we arrive at the operational value of the CEV for one of the armies. Unless the opponent has been ordered not to engage in combat, but to suffer attack without retaliation, we need to obtain the CEV for the enemy returning the attack.

We thus have two CEVs, one for each side. If the two armies are Red and Blue and their CEVs are CEVR and CEVB respectively, the number lost from Red is:

$$\text{Number of Red losses} = \text{CEVB} \times \text{number in Blue army}$$

The number lost from Blue is:

$$\text{Number of Blue losses} = \text{CEVR} \times \text{number in Red army}$$

We calculate these two quantities, then subtract them from the numbers in each army at the beginning of the current round of the engagement. This tells us how many survive to take part in the next round.

This procedure simulates simultaneous combat. The armies do not politely 'take turns' to fire at each other! However, this simple approach to simultaneous combat only applies if the period represented by a turn is relatively short. If it represents a period which is a relatively large part of the total period of combat, the result might be total mutual destruction in one turn. This very rarely happens in real life. Indeed, the combat routine should include provision for the unit routing or surrendering if its numbers fall by excessive amounts, say, by 30% or more.

The use of CEV tables allows a flexible approach to combat resolution. The same routine can be used in varying circumstances in the same game or in different games. In NASEBY we have four CEV tables. One is for combat when units are two squares apart. At that range the pistols of the cavalry are ineffective. The muskets are about half as effective as they are at a range of one square. The second table (Table 2) used in NASEBY is for combat at a range of one square — the engaging units are on adjacent squares. The third and fourth tables cover close combat or *mêlée*. The opposing units occupy the same square and are fighting at close quarters. In these circumstances the swords carried by the cavalry come into play. We have a separate table for the first turn of close combat for, during this turn, the musketeers and cavalry are still able to use their firearms. In second and subsequent turns they will not be able to use these since they are unable to reload them while in *mêlée*. Their CEVs are then related to the effectiveness of the sword, and the butt of a musket being used as a club. This illustrates how a set of CEV tables can be used to model a range of battle stages. We could also devise tables for other situations, such as fighting on different types of terrain or in different weather conditions.

The technique is also applicable to close combat in skirmish wargames. It is used for this purpose in JUNGLE ATTACK. In a skirmish wargame, each unit represents an individual fighter. Each unit begins the game with a value stored as a characteristic of the unit which we will refer to as *combat potential* (CP). CP represents a unit's current ability to engage in close combat. Unless the game requires otherwise, all units begin the game with equal CP. It could be reduced by wounds received prior to engaging in close combat. As a result of a round of combat, treated in the way described above, the CP is reduced by a given amount. We equate this as a loss of combat potential due to injury and fatigue. When the CP level drops below a certain value, the unit is eliminated from the game. In real terms the unit is being either killed, seriously wounded, panicked or becoming so fatigued as to be ineffective for further close combat. Over a suitable period

a unit would recover from fatigue, but almost all skirmish wargames represent only a few minutes of real time, during which, such recovery would be minimal. Thus the routine described in this section is applicable to the resolution of many kinds of combat.

The listing for the combat resolution subroutine RESCOM appears on lines 2860 to 2880 of the JUNGLE ATTACK program (Chapter 6). The first two lines consist of the calculations of the address of the CEV of each unit. These addresses are PEEKed to obtain *cev1* and *cev2* for armies *p1* and *p2*, respectively. The final line of the algorithm calculates the decrease in combat potential of each army (*dcp1*, *dcp2*) by multiplying the combat strength of its opponent by the opponent's CEV.

RESCOM is usually called from a subroutine designed to process a given type of combat. The NASEBY program (lines 4950 to 5220) provides an example. Combat potential is expressed as the number of men in each unit. In the combat phase of the game, the commander of one side selects a unit to attack a selected opposing unit. The rules of the game require that the attacking unit must already be facing the attacked unit. There is actually no provision in the program to ensure that this condition is fulfilled, since we prefer to leave the setting of and compliance with such conditions to the players. This makes the subroutine applicable to other games in which the rules may be different. When the two combatant units have been selected by the commander, using the cursor keys, the COMBAT subroutine calculates their distance apart. If they are more than 2 squares apart, the selection is rejected, as this is greater than the range of the musketeers. If they are one or two squares apart, or are on the same square, COMBAT selects the appropriate CEV table for handling COMBAT at these distances, and finds the CEVs of the two units. If the units are in combat on the same square, *mêlée* conditions apply. There are two CEV tables for *mêlée*, one for the first turn after the attacker enters the square occupied by the defending unit, and one for subsequent turns. This is necessary because, in the first turn, musketeers are able to fire their loaded muskets and cavalry are able to use their pistols. In subsequent turns, there is hand-to-hand fighting with swords or musket-butts. When the correct CEV table has been decided upon, COMBAT then calls RESCOM to calculate the CP (numbers of personnel) lost by each unit. On return to COMBAT, the amount by which each unit is to be decremented is displayed. Players then use the briefing facility to adjust the combat potential of the units concerned. In doing this they may take into account any relevant rules of the game. For example, if the *mêlée* was preceded by a charge into the square, the CP of the occupying unit is further reduced. The same applies if it is attacked on the flank or in the rear. Once again, these factors *could* be taken into account automatically by the computer. We have not provided for this, to allow flexibility in the rules of the games, but readers with programming proficiency could easily add automatic features to COMBAT.

A slightly different approach is taken in the close combat subroutine, CLOSE, of JUNGLE ATTACK. This routine is completely automatic. The routine uses PROX to find squares on which two or more opposing units are present. For every such square found, the subroutine compiles a list of the unit numbers and their CPs (lines 2490–2520). It then displays the

details of the units of each army (lines 2580–2620). Every unit present is considered in turn as an attacking unit. But, for fairness, the army from which the attacking unit is chosen is decided at random. Thus the first unit to attack may belong to either army. The second attacker may belong to the same army or the opposing army. By the end of the routine, all units of both armies will have attacked once each. Having decided on an attacker, the computer selects an opponent unit at random from the enemy units present on the square. RESCOM is then called and the effects of combat are calculated. In this instance, although RESCOM calculates the effects of combat for *both* combatants, the CP of the attacked unit is the only one decremented (automatically) on return to CLOSE. The routine accumulates the total value of CP lost for the armies, as well as the number of units eliminated from either army. At the end of the routine, these particulars are displayed.

Handling wargaming data

5

A computer that is running a wargame program needs access to a large amount of data. It needs to be able to find out the characteristics of every fighting unit, and where it is located. It needs to have all features of the terrain map stored in its memory, as well as information about the terrain itself, such as the amount of cover it provides and the speed with which units are able to move across each region of it. Finally, the computer may need to refer to several tables of data when resolving combat or carrying out other tasks. Another type of data frequently required during the running of a wargame is the machine code for the routines used to speed up certain slow stages in the program.

Fortunately for wargamers, the Amstrad computers have ample memory to spare for storing data. Even so, we must not waste what we have. The most economical way of storing data is to place it directly into blocks of memory specially reserved for the purpose. During a game, data is placed in memory by using the POKE statement. Data already in memory can also be altered in this way, if necessary. Data is read from memory, using the PEEK statement.

The sections of memory used for wargaming data are shown in Figure 5.1. The block begins at address 28824. Memory below this address (about 28 kilobytes) is available for the BASIC program and its variables. The computer is prevented from using the data area for other purposes by the 'MEMORY 28823' statement that appears near the beginning of every wargame program.

The first 176 bytes of the data area are used for storing the patterns of the special graphics characters that we use for representing fighting units on the map. There is room for 22 of these. They are defined by using the SYMBOL statement in a subroutine of the game program.

The next 997 bytes are allocated to machine code. The standard routines contained in the MCODE file (see end of Chapter 3) occupy only 393 bytes, leaving ample room for the reader to store additional routines. This space could also be used for data. The machine code routines are placed in

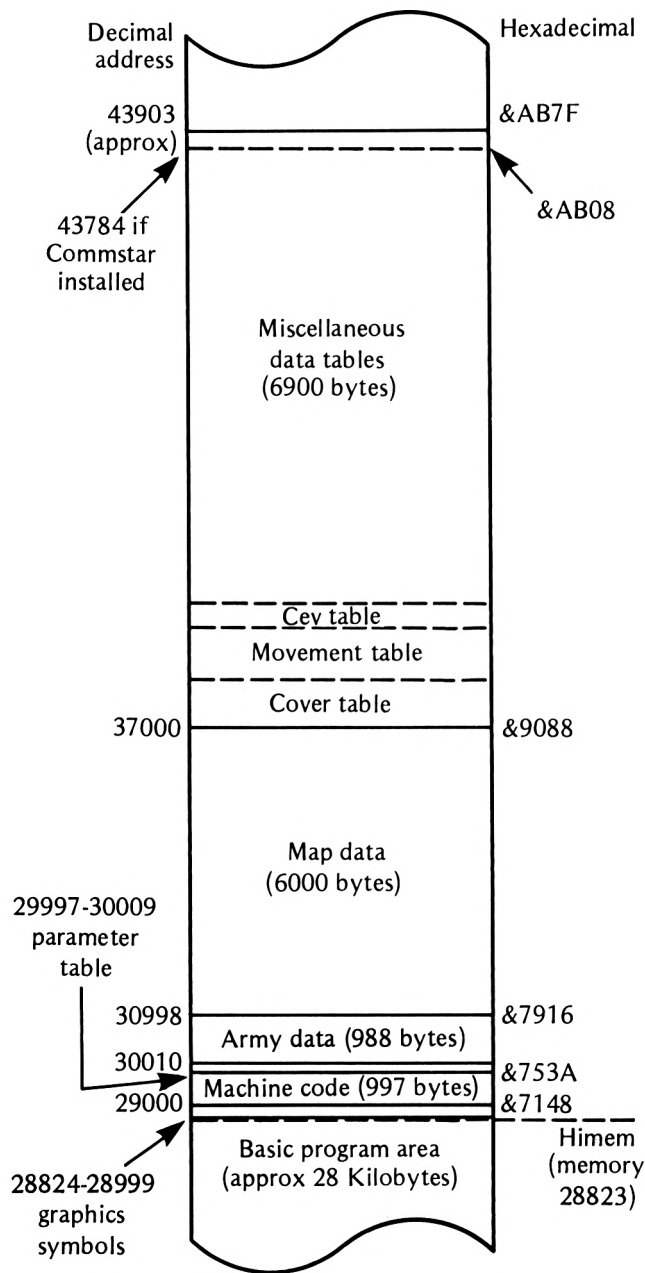


Fig 5.1 Storage of wargame data in Amstrad's memory.

memory by loading the binary file MCODE during the initial stages of the wargame program.

Table 3 shows how we use the block of memory from address 29997 to 30009. The byte at 30000 is incremented at the beginning of each player-

turn. If the game is saved, the number stored here registers the stage the game has reached, ready for the next session of play. The number of armies, stored at 30001 is usually two, but in some games it might be three or four. Many subroutines need to process the data of each army in turn. If the processing algorithm is placed in a loop such as 'FOR j=1 to PEEK(30001)', the routine operates correctly in any wargame program, independently of the number of armies involved.

Table 3 Special data block for wargaming data

Addresses	Description
29997-8	Last address used by data (<i>final</i>)
29999	Number of armies/tables (used in INTELLIGENCE)
30000	The number of the current player-turn
30001	Number of armies
30002/3	Base address of army 1
30004/5	Base address of army 2
30006/7	Base address of army 3
30008/9	Base address of army 4

The values in 30002 to 30009 tell the computer where to find the details of each army. The system allows there to be up to four armies in a game. In order to allow the greatest flexibility in making use of the area reserved for army details, the base addresses of the armies are not fixed. The details of each army are stored in blocks of memory which follow directly after one another, with no gaps between. This allows the total number of units in all armies to be as large as possible. Since the base addresses of armies vary from game to game, it is necessary to have a fixed place in memory to which the computer can refer to find these addresses.

The army base addresses are stored in the usual way for storing addresses in computer memory, in two consecutive bytes. These are referred to as the *low byte* and *high byte*, and stored in that order. The high byte is calculated first:

$$\text{highbyte} = \text{INT}(\text{address}/256)$$

The low byte may then be calculated:

$$\text{lowbyte} = \text{address} - \text{highbyte} * 256$$

These values are then POKEd into the pair of addresses, for example, the low byte into 30002 and the high byte into 30003. The stored address can be reconstituted by PEEKing the addresses to find the values of the low and high bytes. Then the calculation is:

$$\text{Address} = \text{lowbyte} + \text{highbyte} * 256$$

The main data area ranges from 30010 to about 43903, which is over 13 kilobytes. First in this area is the army data, which is allowed a total of 988

bytes. This is enough for storing details of 107 units if there are two armies, 106 units if there are three, and 104 units if there are four. NASEBY, which has rather more units than most computer wargames, has only 62 units. This section should therefore be more than sufficient for almost all wargames programs. The terrain map requires a large amount of memory, since six bytes are needed for each square. The map data occupies a fixed section of memory from 30998 to 36999. 30998 and 30999 hold the number of rows and columns in the map. As explained later, the data for each half-square of the map is stored from 31000 onward. The 6000 bytes reserved for this purpose allows the map to have up to 1000 squares.

Finally, we have approximately 6900 bytes available for storing miscellaneous data. Here we store tables of data such as the cover table (the cover values for each square of the terrain), the movement table and the table of CEVs (see Chapter 4). The address of the last byte used for data is stored in 29997/8. This is used when a game is saved to tape or disk, so that all data relevant to the game is saved, without making the data file unnecessarily long.

Having outlined the system for data storage, and before describing the utilities that the programmer can use for storing data, we look more closely into the storage of certain kinds of data.

Army data

The details of each army are stored in a block of memory starting from a base address held in 30002 to 30009. When the program is run, the army addresses are copied into an 'army address' array, name *aa()*. For example, *aa*(1) holds the address of army 1. The data of each army begins with details that refer to the whole army:

BASE to BASE+7	The army name, in ASCII code (8 bytes), <i>aname\$()</i>
BASE+8	The number of units in the army, <i>units()</i>
BASE+9	Paper number for army symbols, <i>pa()</i>
BASE+10	Pen number for symbols, <i>pe()</i>

These frequently-required values are copied into the arrays named above at the beginning of the game.

The characteristics of each army unit are then stored in consecutive block of 9 bytes, from BASE+11 onward. The base address of each 9-byte block (usually held in a variable *bu*) is calculated by using the equation:

$$bu = aa(na) + 2 + 9 * u$$

where *na* and *u* are the number of the army and unit, respectively.

The contents of each 9-byte block are:

<i>bu</i>	Number of type of unit
<i>bu</i> +1	Row location of unit
<i>bu</i> +2	Column location of unit
<i>bu</i> +3	Symbol number used for displaying unit

bu+4 Status byte
 bu+5 Arms byte
 bu+6 Weapon skill or disorder byte
 bu+7 Combat potential or number of personnel in the unit
 bu+8 Morale byte

The status byte contains data stored in a compact form. Figure 5.2 illustrates the technique of using individual bits as indicators of the current status of the unit. In skirmish games, such as JUNGLE ATTACK, the lower two bits are reserved for storing the activity status of the unit. Two bits can represent values from 0 to 3 and thus indicate activity status, as shown in the figure. In a battle game, in which each unit represents a large number of warriors, these two are used to indicate the direction in which the unit is facing. Directions are coded as 0=north, 1=east, 2=south and 3=west. Bit 2 of the status byte is usually reserved to indicate hidden status in games which have hidden movement (see Chapter 7). Bit 6 indicates panic status, while the other bits are available for other purposes, according to the design of the game.

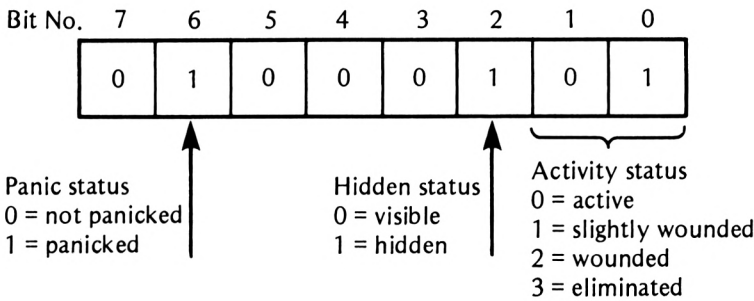


Fig 5.2 How the bits of the status byte are used to store data for skirmish wargames.

The remaining four bytes (*bu+5* to *bu+8*) can be used for storing any kind of numerical data, subject to the restriction that values held in one byte must be in the range 0 to 255. As explained in Chapter 2, certain subroutines expect to find these bytes reserved for special purposes, as indicated in the list above. In any program in which such subroutines are used, the bytes must be allocated accordingly. Otherwise, they are available for the programmer to use as required.

Map data

This is stored from address 30998 onward, 30998 holding the number of rows in the map (*nrows*) and 30999 holding the number of columns (*ncols*). The base address of the data which defines map squares (*map*) is 31000. The squares are defined in two sections, top half and bottom half (see Figure 3.3). The data is listed from the top left corner of the map, running along each *screen* row in turn, down to the bottom right corner. Thus the first data listed is for the top halves of the squares in row 1. This is followed by the

data for the bottom halves of squares in row 1, then the top halves of row 2 and so on. There are three bytes of data for each half-square: paper, pen and character.

Other data

This is held in blocks of memory referred to as data tables. The pattern for storing data tables is the same as that used for storing the map. The first two bytes hold the number of rows and the number of columns in the table. These are followed by a block holding the data itself. Data may be listed by rows (as is the map) or by columns, depending on which is the more convenient for the program to read. The first table begins at address 37000. This is the cover table, which stores the cover value for each square of the map. The next table is the movement table, which holds the movement allowances for each square of the map. The first address of this may be calculated by multiplying the number of rows of the *cover* table by its number of columns and adding 37002 to the product.

The third table is the CEV table (Chapter 4). This is considered as a set of sub-tables, one for each type of combat. The number of rows and columns in each sub-table equals the number of types of unit in the game. The number of rows in the whole table is the same. The number of columns is the number of rows multiplied by the number of sub-tables. After this there may be other tables for various purposes, depending on the nature of the game.

TABLER

In the Key-in way, table data is loaded into memory by the cipher programs. The Utility way uses a special program called TABLER, as follows.

First key in and save the program listed below.

```

10 REM ** TABLER **
20 MEMORY 28999
30 m$="Base address":l=37000:u=43000:GOSUB 330:base=n
  umber
40 CLS:PRINT"Existing data file? (Y/N)":GOSUB 300
50 IF a$="N" THEN 110
60 PRINT:INPUT"File name";f$
70 f$=UPPER$(f$)
80 LOAD f$,base
90 nrows=PEEK(base):ncols=PEEK(base+1)
100 GOTO 130
110 m$="Number of columns":l=1:u=1000:GOSUB 330:ncols
  =number:POKE base+1,ncols
120 m$="Number of rows":l=1:u=INT(6000/ncols):GOSUB 3
  30:nrows=number:POKE base,nrows
130 CLS:FOR j=1 TO nrows:PEN 2:PRINT"Row"j:FOR k=1 TO
  ncols
140 PEN 3:PRINT"Column" k = "TAB(15)PEEK(base+2+(k-1)
  +(j-1)*ncols)" ";:PEN 1
150 v=-1:WHILE v<0 OR v>255

```

```

160 INPUT v$: IF v$="." THEN v=255:ELSE v=VAL(v$)
170 WEND
180 IF v$<> "." THEN POKE base+2+(k-1)+(j-1)*ncols,v
190 NEXT:NEXT
200 CLS:PRINT"Review? (Y/N)":GOSUB 300
210 IF a$="Y" THEN 130
220 CLS:INPUT"File name";f$
230 SAVE f$,B,base,2+ncols*nrows
240 PRINT"Backup? (Y/N)":GOSUB 300
250 IF a$="Y" THEN 220
260 PRINT "File "f$" requires";2+nrows*ncols"bytes"
270 PRINT:PRINT"From"base"to";base+1+nrows*ncols
280 PRINT:PRINT"Tabler finished"
290 LOCATE 1,23:END
300 a$=UPPER$(INKEY$):REM Y/N ***
310 IF a$<>"Y" AND a$<>"N" THEN 300
320 RETURN
330 number=-1000:WHILE number<1 OR number>u:REM SELEC
TNUMBER ***
340 CLS:PRINT m$("1" to "u");
350 INPUT;number$:number=VAL(number$):WEND:RETURN

```

When the program is run, you are first asked to quote the base address. This is the first address of the block of memory at which the data is to be stored. If you are typing in the cover data, for example, the base address is 37000. For other tables you need to calculate the address required and type it in. When calculating, remember to add two extra bytes to allow for storage of the number of rows and columns of the table. Answer 'Y' if you have a data file that you wish to edit, otherwise answer 'N'. If you answer 'Y', you are asked its name. The file is loaded and stored in memory. If you answer 'N', you are asked the number of rows and columns it is to have.

Data is entered by rows. The screen clears and 'Row 1' appears. On the line below 'Column 1 = ' appears, followed by a number, in red. If you are starting a new table, this will probably be zero. If you are editing an existing data file, the number is the value currently stored in row 1, column 1 of the table. The yellow question-mark on the right indicates that the computer is waiting for you to enter a value for row 1, column 1. Type this in and press ENTER. You are then asked to key in row 1, column 2, and so on across the whole of row 1. Note that, if you are editing a table and there are data items that you do not wish to change, it is sufficient to key '.' (full-stop), instead of a number. If you do this the item currently stored for that location in the table remains unchanged.

After the data for all rows has been entered, you are given the chance to review what you have just keyed in. Key 'Y' or 'N' accordingly. If you review the table, the screen displays the stored values row by row and column by column, as above. The red numbers will be those you have just keyed in. Re-key any that you wish to change. To simply check the values, step through the table keying '.' repeatedly. If you key 'N' to the question 'Review?', you are asked for the name under which the data is to be saved. The program saves to tape or disk in the usual way, and you have the opportunity to take back-up copies of the file. When this is complete, the screen displays the number of bytes required by the table, its base address (the one you keyed in at the start of the program) and its last address. Keep

a note of these addresses for future reference. The program is then finished.

How TABLER works

- 20 Protect data memory.
- 30-120 Inputting requirements.
- 130-190 Inputting data.
- 200-210 Option to review.
- 220-250 Saving and back-up.
- 260-290 Displaying storage information.
- 300-320 Subroutine to accept Y/N input.
- 330-350 Subroutine to accept numeric input within limits.

Marshalling games data

Having prepared data files for the armies, the map, and the data tables, the final step is to combine these into the one large data file required by the game program. The program which does this is called INTELLIGENCE, and is listed below.

```

10 REM ** INTELLIGENCE **
20 MEMORY 28999:POKE 29997,58:POKE 29998,117:POKE 299
99,1:POKE 30000,0
30 CLEAR:s=PEEK(29997)+256*PEEK(29998):na=PEEK(29999)
:CLS:PRINT"Army name";na;:INPUT;file$
40 file$=UPPER$(file$)
50 OPENIN file$
60 INPUT #9,types,units,ndetails,sp,pa,pe
70 DIM units(types),fig(units),rsp(units),csp(units),
details(units,5):rsp(1)=0
80 FOR k=1 TO types:INPUT #9,units(k):NEXT
90 FOR k=1 TO units:INPUT #9,fig(k):NEXT
100 IF NOT sp THEN 130
110 FOR k=1 TO units:INPUT #9,rsp(k):NEXT
120 FOR k=1 TO units:INPUT #9,csp(k):NEXT
130 IF ndetails=0 THEN 160
140 FOR k=1 TO units:FOR d=1 TO ndetails
150 INPUT #9,details(k,d):NEXT:NEXT
160 CLOSEIN
170 PRINT"DATA LOADED - BEING STORED"
180 POKE 30001+na*2,INT(s/256)
190 POKE 30000+na*2,s-PEEK(30001+na*2)*256
200 length=LEN(file$):FOR j=0 TO length-1:POKE j+s,AS
C(MID$(file$,j+1,1)):NEXT
210 s=s+length
220 WHILE length<8:POKE s,32:length=length+1:s=s+1:WE
ND
230 POKE s,units:s=s+1:POKE s,pa:s=s+1
240 POKE s,pe:s=s+1
250 FOR k=1 TO units:count=1:cum=0
260 WHILE cum<k:cum=cum+units(count)
270 count=count+1:WEND
280 POKE s,count-1:s=s+1:POKE s,rsp(k):s=s+1
290 POKE s,csp(k):s=s+1:POKE s,fig(k):s=s+1
300 FOR d=1 TO 5:POKE s,details(k,d):s=s+1:NEXT:NEXT

```

```

310 POKE 29998,INT(s/256):POKE 29997,s-PEEK(29998)*25
6
320 na=na+1:POKE 29999,na:IF na<3 THEN 30
330 IF na=5 THEN 360
340 PRINT"Another army? (Y/N) ";:GOSUB 580
350 IF a$="Y" THEN 30
360 POKE 30001,na-1
370 CLS:INPUT "Map name";file$
380 file$=UPPER$(file$)
390 LOAD file$
400 final=30999+PEEK(30998)*PEEK(30999)
410 PRINT"Data tables? (Y/N) ";:GOSUB 580
420 IF a$="N" THEN 520
430 INPUT "Table name";file$:file$=UPPER$(file$)
440 base=0:WHILE base<37000 OR base>43000:INPUT "Base
address";base:WEND
450 LOAD file$,base
460 CLS:PRINT"Table starts at ";base
470 tableend=base+PEEK(base)*PEEK(base+1)+1
480 PRINT"Table finishes at ";tableend
490 final=MAX(tableend,final):POKE 29998,INT(final/25
6):POKE 29997,final-PEEK(29998)*256
500 PRINT"Another table? Y/N":GOSUB 580
510 IF a$="Y" THEN 430
520 INPUT "Name of game";file$:file$=UPPER$(file$)
530 file$=file$+"D"
540 SAVE file$,B,29997,final-29996
550 PRINT"Back-up? Y/N":GOSUB 580
560 IF a$="Y" THEN 540
570 END
580 a$=UPPER$(INKEY$):IF a$<>"Y" AND a$<>"N" THEN 580
:REM Y/N ***
590 RETURN

```

Before running the program, make a list of all the files you will require and make sure that they are ready on disk or tape. The file that you are about to prepare will hold eight kilobytes or more of data. If you are using disks, make sure that the disk has room to take a file of this length. The procedure for loading each of the army, map and data table files is always the same, so we describe it once before outlining the sequence of the program. At each stage of the program you will be asked the name of the file to be loaded. Check that the tape is in the recorder, wound to the start of the file concerned or, with disks, that the correct disk is ready in the drive. Then key the file name and press ENTER. With tapes, press PLAY and any other key when requested. The data is then loaded into the correct part of memory. With disks, loading is automatic.

When all is ready, run INTELLIGENCE. It first deals with armies. You are asked the file name of the first army. Follow the procedure described above to load the data of the first army. After this you are asked if you wish to load another army. Key 'Y' and load the details of the second army. You will then be asked again if you wish for another army. Key 'Y' or 'N'. You are allowed to have up to four armies, provided that the total number of units is not greatly in excess of 100 (see section on army data, above). The map is loaded next, using the same procedure. After this, you are asked if you wish to load data tables. Answer 'Y' or 'N'. If you answer 'Y', you then key in the file name of the table, and its base address. Refer to the list of addresses you made when using TABLER and key in the appropriate base

address for each table. However, if you wish, the table data can be loaded to a different base address. This allows you to use tables that were originally prepared for use with another program. When the table is loaded, the screen tells you the base and last address of the table. You are invited to load more tables, unless the last address used exceeds 43000, in which case there is no memory left for further tables.

When you have finished entering tables (or if you do not require tables), you are asked to key in the name of the game for which the data is to be used. The name is not necessarily the name of the game itself, but the name under which the game *program* is saved. For example, the game program of JUNGLE ATTACK is saved as "ATTACK" and this is the file name to be typed in. The complete set of data is then saved under a file name which consists of the name of the game with a 'D' appended to it. For example, if the game is called ATTACK, the data file is called "ATTACKD". You are given the chance of saving back-up copies, after which INTELLIGENCE ends.

How INTELLIGENCE works

- 20 Protecting data memory, putting the start address (30010) and number (1) of the first army into memory. Set TURN to 0.
- 30 Clearing memory, then reading the start address and number of the first army from memory; input army name.
- 40–160 Loading army data file into arrays.
- 170–320 Transferring data from arrays to memory; updating army start address and army number.
- 330–360 Inviting another army unless four already stored.
- 370–400 Load map data and calculate last address used (*final*).
- 410–490 Loading tables, and updating *final*.
- 500–510 Inviting another table.
- 520–570 Saving and back-up.
- 580–590 Subroutine to accept Y/N input.

Jungle Attack

6

This chapter describes how to set up and play a skirmish wargame, based on jungle combat on a Pacific island during World War 2. If you wish, you may build a model of the terrain, and play the game with model soldiers, using the computer as an aid to play. Alternatively, you may play the game entirely on the computer, without using models of any kind.

A skirmish wargame is one in which each unit in an army, or symbol on the terrain map, represents one soldier on the battlefield. If you are using models, you require one model for each unit, a total of 37 models. Figures 6.1 and 6.2 are designs for cut-out cardboard model soldiers, that you can copy if you prefer not to buy models.

Each soldier has individual characteristics, such as the weapons carried and the skill in using them. Soldiers are moved individually and may panic, be wounded or be killed in action during the course of the game. We have more to say about this later, but first we set the scene for the skirmish. The terrain resembles one of our favourite tropical beaches. This is on the island of New Guinea close to the Indonesian town of Jayapura. The beach is



Fig 6.1 Australian infantryman.



Fig 6.2 Japanese infantryman.

known locally as 'Base G' for it is here that Allied forces landed in 1945 in their advances against the Japanese.

Scenario

The year is 1945 and the Allied offensive in the Pacific is gaining momentum. On some of the islands, there are still pockets of Japanese troops fiercely determined to hold on for as long as possible. One such group has established itself on a high rocky outcrop overlooking one of the few serviceable roads through the jungle (Figure 1.1). From this vantage point they are able to prevent the westward Australian advance along this road. Australian troops, attempting to approach along the road to dislodge the Japanese, have met a deluge of fire and have been forced to retreat. As a result of this, the Australians have decided to try a different tactic. A party of 10 men is to land on the beach close to the rocky outcrop. They are to disappear as quickly as possible into the jungle, which borders on the beach. From there they are to try to scale the rocky outcrop and attack the Japanese at close quarters. The Japanese have the advantages of being in good cover, being on higher ground, and of outnumbering the Australian party. In addition, they have high morale and will fight to the last. However, a second party of 10 Australian troops is on its way to the scene, arriving either by sea or by road. Among them are soldiers especially trained in close combat, to take part in the final stages of the attack.

The aim of the Japanese is to beat off the Australian attack, yet still to have sufficient troops to retain control of the road. The aim of the Australians is to reduce the Japanese party to an ineffective state.

The terrain and movement

The terrain (see map, Figure 1.1) consists mainly of jungle. This is displayed in dark green on the screen, with black 'tree' symbols scattered over it. At the south end of the area are the sea (blue) and beach (bright yellow). To the west are the two rocky outcrops (bright green), the northern one of which is occupied by the Japanese. The summits of these outcrops (light green) are open ground, as are the areas either side of the main road (red). To the east is an area of swamp (pastel blue). The jungle and swamp are covered with a network of narrow tracks (yellow).

The side of a square on the map represents a distance of 33m. The distance that a unit may move in each turn depends on the type of terrain the unit is crossing. When you move a unit, the computer automatically takes all factors into account and will not let you move a unit more than the permitted distance. The figures below are intended just to help you plan your moves. The maximum distance you can move is calculated by the computer, as follows. Each unit has a movement allowance of six points per turn. When a unit moves one square, it expends these points at the following rates:

<i>Terrain type</i>	<i>Movement cost (per square)</i>
Road	1
Open ground, track	2
Beach	3
Rocky outcrop, swamp or jungle	4

Units which are wounded have their movement allowance reduced to four points. Units which are in a state of panic may not move.

The areas of the map provide different degrees of cover against enemy fire. The road, open ground, swamp, and beach provide no cover (level 0, see Chapter 4). Other areas provide light cover (level 1). Since jungle vegetation overhangs the marsh, the squares at the edge of the marsh also provide light cover. The game allows units to 'dig in'. This gives the unit concerned heavy cover (level 2) for as long as that unit remains on the square on which it has dug in.

The armies

Army 1, the Australians, consists of the following units, all with high morale:

The first party to land on the beach:

Sergeant (1), armed with a sub-machine-gun and two grenades. High weapon-skill and close combat-skill.

Privates (8), each armed with a rifle and two grenades. High weapon-skill, but weak at close combat.

Private (1), armed with a light machine-gun, very high weapon-skill and close combat-skill.

The second party:

Corporal (1), moderate weapon-skill, armed with a sub-machine-gun and two grenades. Moderate weapon-skill, but good at close combat.

Privates (8), each armed with rifles and two grenades. Moderate weapon skill, but highly-trained at close combat.

Private (1), armed with a mortar. High weapon-skill, but poor at close combat.

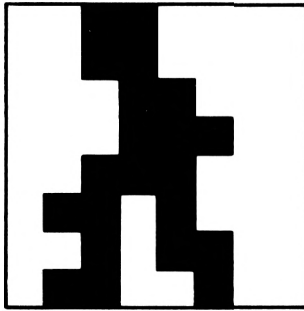
The symbols used to represent the Australian units are shown in Figure 6.3.

Army 2, the Japanese, consists of the following units, all of which have the highest possible morale and all of which are dug in at the start of the game.

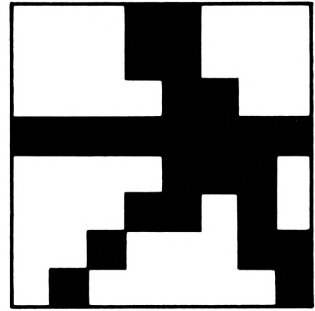
Sergeants (3), each armed with a sub-machine-gun and two grenades. Very high weapon-skill and close combat-skill.

Privates (4), each armed with a rifle and three grenades. Moderate weapon-skill and close combat-skill.

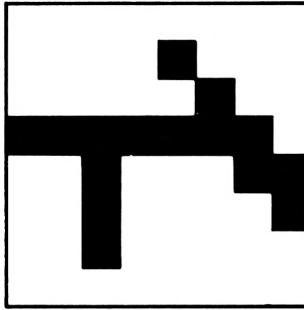
Privates (4), each armed with a light machine-gun. Crack shots and excellent at close combat.



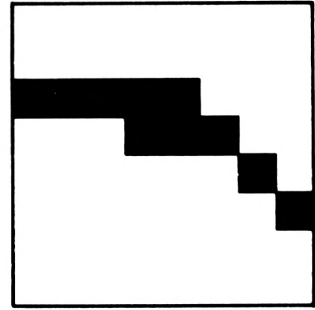
234



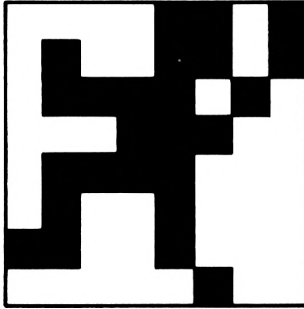
235



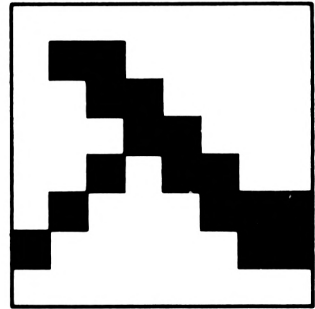
236



237



238

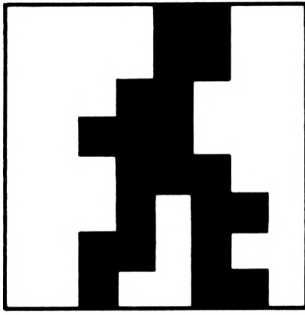


239

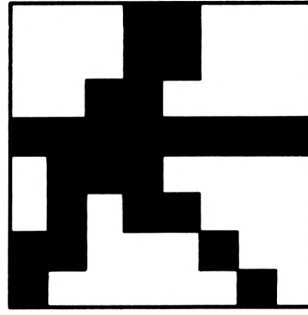
Fig 6.3 Symbols used for Australian units, displayed white on red. Numbers are the code under which they are defined: 234, sergeant with sub-machine-gun; 235, private with rifle; 236, private with light machine-gun; 237, corporal with sub-machine gun; 238, private throwing grenade; 239, private with mortar.

Privates (4), each armed with a rifle and three grenades. Poor shots, but excellent at close combat.

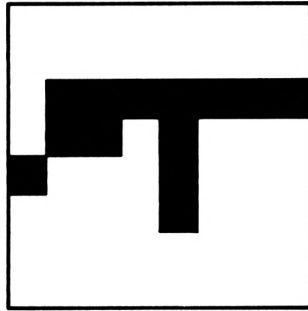
Privates (2), each with a mortar. Of high weapon-skill but moderate close combat skill.



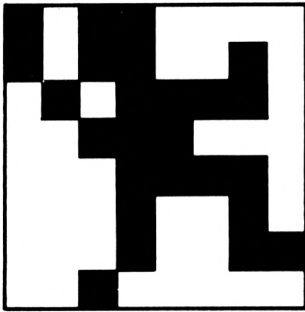
240



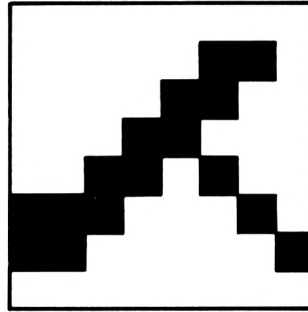
241



242



243



244

Fig 6.4 Symbols used for Japanese units, displayed yellow on blue. 240, sergeant with sub-machine-gun; 241, private with rifle; 242, private with light machine-gun; 243, private throwing grenade; 244, private with mortar.

The symbols used to represent the Japanese units are shown in Figure 6.4.

The status of each unit may change during the game. At the beginning, the first party of Australians and all Japanese are active. If a unit becomes wounded, movement is reduced and firing is less accurate. If a unit is wounded twice, it is counted as killed. If a unit is killed it takes no further part in the game. A unit may panic if it is fired at, but is not hit. A panicking

unit is not able to move or fire *during its next turn*. At the end of its next turn it is automatically rallied, and it is then able to move and fire normally. These rules are all taken account of automatically by the computer.

The weapons

The majority of units have rifles, maximum range nine squares. Sub-machine-guns are more effective at close range but their maximum range is only four squares. The light machine-guns are the most effective, with a maximum range of 18 squares.

Grenades may be thrown a distance of one or two squares and are highly effective in the square in which they land. An important advantage of grenades is that they can be thrown *over* a 'cover' square. For example, from a square just outside a jungle area, a grenade may be thrown either one square *or two squares* into the jungle. By contrast, rifle or machine-gun fire cannot penetrate further than one square into the jungle. As a general rule, rifle and machine-gun fire can pass out of and into, but not *through*, a 'cover' square.

It is considered that all units have sufficient ammunition for firing rifles and machine-guns every turn of the game, but the number of grenades carried is limited.

Mortars have a much longer range than the other weapons. In this game they can hit any square of the terrain, from any location within the terrain. Like grenades, mortar bombs can be fired over intervening 'cover' squares, and burst in a 'cover' square. The mortar bomb is effective not only in its target square but also, to a lesser extent, in each surrounding square. The mortar is thus an extremely powerful weapon. However, there are restrictions on its use. The mortar unit represents one soldier armed with a mortar, but a mortar requires *two* men to fire it. Thus the mortar cannot be fired unless one other unit is present on the same square. If the mortar private is wounded, the mortar is fired with reduced accuracy. The mortar cannot be fired if the private is panicked or killed. Once again, all these rules are taken care of by the computer. An additional rule, which is not part of the program, but which players may find gives a fairer balance, is that the Japanese may only fire one of their mortars (the first one deployed). The unused 'mortar' unit may be deployed as an infantryman armed with a rifle. This also releases the assistant unit for other combat. The reason for having two mortars on the Japanese side is to give them extra fire-power for the version of JUNGLE ATTACK which uses hidden movement (see Chapter 7).

The game

One player takes command of the Australian army, the other takes command of the Japanese army. The Australian commander has first turn. At all stages of the game, any number of units may be present on the same square.

The way to use the computer to deploy, move or fire units is described in the next section. In summary, the stages of the game are:

Deploy Australian troops The Australian units of the first party are to be deployed on the seaward edge of the beach (row 23), where they have just landed.

Deploy Japanese troops The Japanese units are to be deployed in the open ground on the northern rocky outcrop. Now follow a series of up to 20 game-turns, each game-turn consisting of two player-turns, played by the Australian and the Japanese players. Each player-turn consists of the following phases.

Advance phase Any one or more units may be advanced, any number of squares, up to its limits. Instead of advancing, a unit may be instructed to dig in.

Advance/Fire phase Units may be advanced again, or may fire their weapons. Units with grenades may throw them instead of firing.

Close combat phase If one or more units of opposing sides are present on the same square. The outcome depends on close combat skill and is resolved by the computer.

Rally phase Units which were panicked when fired at during the opponent's previous turn are rallied automatically.

At the Australian player's eighth turn, there is an additional deployment phase. The newly arriving units may be deployed on the beach (row 23) or on the road and open ground at the eastern edge of the area (columns 19 and 20) according to the preference of the Australian player. Or the force may be deployed partly in one area and partly in the other. The game ends after the twentieth turn, or earlier if one side achieves its aims.

The Australian army wins when one or both of the Japanese mortars (according to the deployment used) are put out of action and fewer than six active or wounded Japanese units remain.

The Japanese army wins when the Australian army is reduced to fewer than 10 active or wounded units, provided that at least one of the Japanese mortars remains in action.

Any other outcome is best taken as indecisive, for there is no true victor. However, players may decide that a 'score' based on the numbers of active and wounded units may be used to decide the 'winner'. Note that the computer does not end the game automatically when one of the victory conditions is achieved. This allows players to agree on any other set of victory conditions that they prefer. Similarly the computer does not insist on troops being deployed according to the rules stated above. If players wish to invent a new scenario for initial deployment positions, tactical aims and victory conditions they are able to use this game program without making any alterations to it.

The game takes several hours to play. At the end of each player-turn you are given the option of saving the game to tape or disk, to be resumed on later occasion.

Computer control

This is the first wargame in the book, so we describe the method of controlling the computer in detail. You will probably need to refer to this section again when playing the other games in the book.

During most phases of the game, the screen is divided into five areas. The top 22 lines display a map of part of the terrain. The bottom three lines are used separately for displaying messages and information, except on the extreme right. There we see the letters "NS EW", a pair of numbers and the letter "F". During a deployment phase, "R" and "+" are displayed in addition.

Most aspects of controlling the troops rely on the cursor. This appears on the map as a flashing patch of contrasting colours. The cursor is moved around the map by using the cursor control keys (as used when you are editing a program line).

The screen displays an area of terrain, 20 columns from west to east and 11 rows from north to south. The numbers displayed at the bottom right corner change as the cursor is moved, telling you which map square the cursor is in. The cursor moves so that it occupies the bottom half of the square it is in (Figure 3.3). If a unit is present in a square, its symbol occupies the top half.

With the cursor below the unit symbol, you can press:

'R' to obtain a *roster* (see below)

'A' to *advance* a unit

'M' to fire a *mortar*

'G' to throw a *grenade*

'D' to command the unit to *dig in*

or

'Space bar' which is used for several different purposes as described below.

Information about the unit's present condition then appears on the bottom line of the screen. This always includes the symbol of the unit, by which you can identify it. In JUNGLE ATTACK, this information also includes the number of grenades carried (green figure), the unit's morale level (blue figure), whether it is dug in or not (yellow letter D, if dug in) and its current status (Active, Panicking, Wounded, or Killed).

Roster Obtaining a roster by pressing key 'R' is an easy way of finding out which units are present on a square. The details of every unit of *your* army present on that square is displayed. The roster lists the units in order, starting from unit 1. There is a short 'beep' as each unit's details are displayed. If there are two or more units with identical details on the same square, the display appears not to change, but the number of beeps indicates how many units of that kind are present. When the details of all units have been displayed, a message 'Roster complete' is shown.

The descriptions of the commands described below apply when there is only one unit on the square. If there are more than one, use the roster command first, as explained shortly.

Advance If you intend to advance a unit, move the cursor to its square and press 'A'. The border of the screen turns dark blue to indicate that you are in 'Advance' mode. The next step is to indicate to where the unit is to move. Move the cursor square-by-square along the route you wish the unit to take. When you reach the final square, press 'A' again. The unit symbol

then moves to the new square. While you are moving the unit, the computer checks how many movement points you have used. If you try to move too far, the message 'Too far' appears and the cursor is returned to the position of the unit symbol. You can then try again. If you try to advance a panicked unit a message will inform you that the unit is panicking and not able to move. Units may be moved only once each during a phase. A message will tell you if a unit has already moved. Note that there may be certain types of terrain (such as the sea in JUNGLE ATTACK) onto which the computer does not let you move. When a legal move has been made, the border of the screen becomes grey again.

Fire If you wish to fire weapons or throw grenades, move the cursor beneath the symbol of the firing unit and press the space bar, the border of the screen turns red. This indicates that you are in 'Firing' mode. Move the cursor to the target square, the square on which the enemy unit is situated. Press the space-bar again, to indicate that you wish the unit to fire its rifle or machine-gun. To make it throw a grenade, press key 'G'. The border turns yellow at this stage while the computer calculates the effect of fire. After this, a message is displayed to tell you the result and the border changes to grey again. If the enemy unit is killed, its symbol disappears from the screen.

Dig in If you intend a unit to dig in, move the cursor to the unit and press key 'D'. If a unit is already dug in, as are all Japanese units at the beginning of the game, pressing 'D' returns it to normal (not dug in).

Two or more units When the 'A', 'D' or space bar keys are used as described above and there is more than one unit on the square, the unit affected is the first one of the roster. If you want to move all the units on the square, simply repeat the procedure above. However, if you want to move, fire or dig-in a unit that is *not* the first one to be shown on the roster, use the roster to select the unit you require. Move the cursor to the square and press 'R'. Watch as the roster is displayed and, as soon as you see the details of the unit you require, press the 'A', space bar or 'D' key. Rostering then ceases and you enter 'Advance' or 'Fire' phases or the unit becomes dug in.

It follows from the description above that you can only fire at, or move to, a square that is already displayed on the screen. If you wish to fire at, or move to, a square that is beyond the edge of the screen (assuming that the weapon range or movement allowances allow you to fire or move that far), it is necessary to change the map display *before* firing or moving. To change the display, move the cursor down to the bottom of the screen. When it is moved beyond the bottom of the displayed map, it reappears in the area at bottom right of screen. It can then be moved on to one of the letters "NSEW". If you wish to display the map section to the south of the displayed section, for example, move the cursor on to the "S" and press 'fire'. The new section is then displayed. There is overlap between sections; the new section is six rows to the south of the previous section. In this game, the terrain is only 20 squares wide, which is the same width as the screen, so the whole width of it is displayed. In other games, you can use "E" and "W" to change to sections to the east or west of the current section. In such cases the new section is 10 columns to the east or west.

Mortar Firing a mortar has its own special routine, since its range extends over the whole terrain. Do not move the cursor to the square on which the mortar is situated, but move it to the target square. Before doing this, you may need to change the map section, as described above. Now press key 'M'. The mortar is fired, the border of the screen becomes yellow and the results are reported on the screen. If you are the Japanese player, with two active mortars, it is possible to do this twice in each Advance/Fire phase. The mortar of unit 16 is fired first, that of unit 17 is fired second. However, as explained earlier, players may agree that only one mortar is to be fired at each turn.

Finish The other letters in the bottom right corner have special functions. Letter 'F' is used to indicate when you have finished playing a phase. Move the cursor on to the 'F' and press 'Fire'. The phase ends and the next phase begins. Letter 'R' and symbol '+' appear only during a deployment phase. Their use is described below.

Special instructions

The instructions above cover the main features of game control. Now we run through the phases of JUNGLE ATTACK, to show you what to do at each stage. Some of these instructions may also apply to other games.

1 *Run the program* Answer 'Y' when asked if this is a new game. If you are using a disk drive, data then loads automatically. If you are using a tape recorder, press 'Play' and any keyboard key when asked to do so. The message 'Loading data' appears. As soon as loading is finished, the northernmost section of the map appears on the screen.

2 *Deploy Australian army* The bottom line of the screen shows details of the first unit to be deployed (the Australian sergeant). Change map section (heading south!), until the beach is visible. Move the cursor to the square on which you wish to deploy the sergeant, and 'Fire'. His symbol appears on the square immediately above the cursor. The bottom line of the screen then changes to show details of unit 2, the first of the privates. Position the cursor where required and press 'Fire' for each unit in turn until the message "All deployed" appears. If you are satisfied with the deployment, move the cursor to the 'F' at the bottom right of screen and press 'Fire'. This ends the phase. If you wish to re-deploy one or more units, move the cursor to 'R' and then 'Fire'. You are then allowed to deploy all units again. For any units which are already in a satisfactory position, move the cursor to the '+' and press 'Fire'. This steps the program on, leaving the unit in its existing position.

3 *Deploy Japanese army* The procedure is exactly as at step 2.

4 *Advance phase, Australian army* In this phase you may advance as many units as you wish, once each, as already described. There is no digging in during this phase. You can change map sections as often as you like between moving units. When you have finished, move the cursor to the 'F' and press the space bar.

5 *Advance/Fire phase* You may move each unit again, as at Advance

phase, you may fire it, as described earlier, or you may order it to dig in. Digging in gives the advantage of hard cover. To dig in, move the cursor to the unit concerned and press 'D'. Unit details change to show the 'dug in' status. A unit which is dug in immediately gains the advantage of hard cover. If you subsequently move a unit which is dug in, it loses this protection. Although the computer does not prevent it, a unit should not dig in when it is on a road square.

Units which are engaged in close combat should not be fired during this phase. Neither should other units fire into a square on which close combat is occurring.

You can change the map section as often as you wish. When you have finished moving, firing and digging in move the cursor to the 'F' and press the space bar.

6 Close combat phase This only occurs when there are units of opposing armies on the same square. The computer detects such occurrences and resolves the combat automatically. The results are presented on the screen. The screen clears and the row and column of the square are reported. A list of the army units appears, together with their details. The total combat potential of each side at the end of this round of close combat is displayed. The rate of fall of combat potential is usually fast at first but is less later, as the units become exhausted. When the CP of a unit falls below one, it is considered to be killed, severely wounded or completely exhausted. Such a unit has its symbol displayed on the screen and is then eliminated from the game.

7 End of turn The screen clears and you are shown a roster of all units of the army which has just completed its turn. Then you are given the option of continuing with the next player-turn, or saving the present state of the game. If you elect to continue, this is the point at which the players change seats, particularly if you are playing with hidden movement (see Chapter 7). The game continues from stage 4 above, except that it is now the Japanese player's turn.

When you wish to finish, if you are saving to tape, place a new tape in the recorder. Then key in the name under which the game is to be saved. Press the 'RECORD' and 'PLAY' keys when asked to do so. Then press any keyboard key.

If you are saving to disk, it is better to save the game on to the same disk as the game program. Take care not to use "ATTACKD" as the file name for the saved game. If you do, the data file which holds the original state of the game is overwritten by the latest data. When saving is finished, the game ends.

8 If you are continuing with the game, the sequence above (steps 4 to 7) is repeated until the end of the Japanese player's turn at game-turn 20 At this stage each player has had 20 turns. You may end the game earlier than this if victory conditions occur. There is an additional deployment phase for the Australian player at the beginning of turn 8. Deploy units 11 to 20 as in step 2 above.

9 When you are resuming a saved game, run the program and answer 'N' when asked if this is a new game You are then asked to type in its file name. With tape, place the appropriate tape in the recorder. Press 'PLAY' and any

keyboard key when requested. With a disk, provided the disk holds the saved data, loading is automatic.

The resumed game begins from the point at which the game was saved. There is no deployment phase.

Morale

Morale is often an important factor in winning or losing battles. The units in this game all have high morale and, to simplify the game, we are assuming that this does not change appreciably during the two minutes of real time that the game represents. For this reason, the program does not take account of morale, even though it is displayed in the unit details. In Chapter 8 we describe a routine which allows details such as morale to be changed by the players during the game. In Chapter 8 you will be shown how to add this routine to JUNGLE ATTACK. Some rules will be suggested to take morale into account in the game.

Setting up

Type in the game program below, and save it under the name "ATTACK":

```

10 REM ** JUNGLE ATTACK **
20 PAPER 0:INK 0,1:PEN 1:INK 1,24:CLS
30 MEMORY 28823:CLS:GOSUB 3030:gturn=2:cevf=100:units
(1)=10:REM START
40 IF PEEK(30000) THEN 70
50 numarmy=1:na=1:dst=1:dfin=10:GOSUB 1300: REM DEPLO
Y
60 numarmy=2:na=2:dst=1:dfin=units(2):GOSUB 1300: REM
DEPLOY
70 turn=PEEK(30000)+1:POKE 30000,turn
80 IF turn>14 THEN units(1)=20:ELSE units(1)=10
90 IF turn=15 THEN numarmy=1:na=1:dst=11:dfin=units(1
):GOSUB 1300: REM DEPLOY
100 numarmy=turn-INT((turn-1)/PEEK(30001))*PEEK(30001
)
110 na=numarmy:fp=0:m$="Advance":GOSUB 2060:REM PHASE
120 na=numarmy:fp=1:mor=1:m$="A/Fire":GOSUB 2060:REM
PHASE
130 p1=1:p2=2:GOSUB 2390:REM CLOSE
140 CALL 29100,aa(na)+15,aa(na)+15,units(na),9,9,2916
5:REM RALLY
150 GOSUB 2890:REM TURNEND
160 IF turn=41 THEN 190
170 IF mo=240 THEN 70
180 GOSUB 3000:REM SAVE
190 PAPER 0:PEN 1:MODE 1:LOCATE 2,2:PRINT"Game finish
ed"
200 GOTO 200
1000 all=-1:tf=0:REM ADVANCE ***
1010 GOSUB 4380:IF mo<>0 THEN GOSUB 1180:REM MOVE/PIP
1020 IF tf=1 THEN 1040
1030 IF mo=97 THEN PRINT CHR$(7):GOTO 1160
1040 GOSUB 1660:REM CURSOR0

```

```

1050 IF mo<244 AND mo>239 THEN tf=0
1060 IF mo=0 THEN 1010
1070 IF d(na,u) THEN f$="Already moved":GOSUB 3960:RE
TURN:REM MESSAGE
1080 pb=PEEK(aa(na)+6+u*9) AND 64:IF pb=64 THEN f$="U
nit panicking":GOSUB 3960:RETURN
1090 pb=PEEK(aa(na)+6+u*9) AND 3:IF pb=3 THEN f$="Uni
t eliminated":GOSUB 3960:RETURN
1100 IF all>-1 THEN 1130
1110 all=6
1120 IF pb=2 THEN all=all-2
1130 all=all-PEEK(amap+xm-1+ncols*(ym-1))
1140 IF all<0 THEN f$="Too far":GOSUB 3960:xn=xp:yn=y
p+1:tf=1
1150 xc=xn:yc=yn:GOTO 1010
1160 IF xf=xm AND yf=ym THEN RETURN
1170 mf=1:GOSUB 3340:d(na,u)=1:RETURN
1180 SOUND 1,60,1:RETURN:REM PIP ***
1190 SOUND 1,956,50:RETURN:REM BOOM ***
1200 FOR j=1 TO PEEK(30001):REM DISPARMY ***
1210 bu=aa(j)+1:PEN pe(j):PAPER pa(j):FOR k=1 TO uni
ts(j)
1220 IF (PEEK(bu+4) AND 3)=3 THEN 1280
1230 IF (PEEK(bu+4) AND 4) AND j<>na THEN 1280
1240 IF (PEEK(bu+4) AND 8) THEN 1280
1250 xs=PEEK(bu+2)-1ft+1:IF xs<1 OR xs>20 THEN 1280

1260 ys=(PEEK(bu+1)-top)*2+1:IF ys<1 OR ys>21 THEN 12
80
1270 LOCATE xs,ys:PRINT CHR$(PEEK(bu+3));
1280 bu=bu+9:NEXT:NEXT
1290 RETURN
1300 dp=1:GOSUB 1950:REM INITIAL:REM DEPLOY ***
1310 CLS #1:CLS #2:PRINT #1,"Deploy "aname$(numarmy);
1320 ud=dst
1330 ny=numarmy:u=ud:scr=3:vx=1:vy=1:GOSUB 1470:REM U
NITLINE/ATTACK
1340 GOSUB 1580: REM COMMAND
1350 IF wind THEN 1400
1360 LOCATE xc,yc-1:PEN pe(ny):PAPER pa(ny):PRINT CHR
$(PEEK(bu+3));
1370 xf=PEEK(bu+2):yf=PEEK(bu+1):xp=xf-1ft+1:yp=2*(yf
-top)+1
1380 IF xf<1ft OR xf>1ft+19 OR yf<top OR yf>top+10 TH
EN mf=0:ELSE mf=1
1390 GOSUB 3340:GOTO 1420: REM MENDIT
1400 IF xc4=4 AND yc4=3 THEN 1420
1410 GOSUB 2000:GOTO 1330:REM NEWMAP
1420 ud=ud+1:IF ud<dfin+1 THEN 1330
1430 CLS #3:f$="All deployed":GOSUB 3960: REM MESSAGE

1440 mo=0:WHILE yc4=1 OR (xc4<>2 AND xc4<>3):GOSUB 15
80:WEND:REM COMMAND
1450 IF xc4=2 THEN 1300
1460 CLS:CLS #1:CLS #2:CLS #3:CLS #4::dp=0:RETURN
1470 bu=aa(ny)+9*u+2:REM UNITLINE/ATTACK ***
1480 PEN #scr,pe(ny):PAPER #scr,pa(ny):LOCATE #scr,vx
,vy:PRINT #scr,CHR$(PEEK(bu+3));
1490 PEN #scr,12:PAPER #scr,11:PRINT #scr,MID$(STR$(P
EEK(bu+5) AND 3),2);
1500 SOUND 1,100,6
1510 PEN #scr,10:PRINT #scr,MID$(STR$(PEEK(bu+8)),2);

1520 IF PEEK(bu+4)>127 THEN PEN #scr,1:PRINT #scr,"D"
;
1530 PEN #scr,4:pbu=PEEK(bu+4) AND 3:IF pbu=3 THEN PR
INT #scr," Killed ";:RETURN
1540 IF pbu=2 THEN PRINT #scr," Wounded ";:RETURN

```



```

1550 IF (PEEK(bu+4) AND 64) THEN PRINT #scr,"Panicked
";:RETURN
1560 PRINT #scr," Active ";
1570 RETURN
1580 CLS #4:PRINT #4,"NS EW":LOCATE #4,1,3:IF dp THEN
PRINT #4," RF+":ELSE PRINT #4," F":REM COMMAND ***

1590 PEN 4:PAPER 7
1600 mo=0:WHILE mo=0 OR (mo>239 AND mo<244):GOSUB 43
80:IF mo<>0 THEN GOSUB 1180
1610 IF wind=0 AND mo=241 AND yc=22 THEN wind=4:mo=0

1620 IF wind AND mo=240 AND yc=1 THEN wind=0:mo=0

1630 IF wind THEN GOSUB 1770:ELSE GOSUB 1660:REM CURS
OR 4/0
1640 WEND
1650 PRINT CHR$(7):RETURN
1660 LOCATE xc,yc:CALL &BB8A:t=TIME:WHILE TIME<t+20:W
END:REM CURSOR0 ***
1670 IF mo=242 AND xc>1 THEN xn=xc-1
1680 IF mo=243 AND xc<20 THEN xn=xc+1
1690 IF mo=240 AND yc>2 THEN yn=yc-2
1700 IF mo=241 AND yc<22 THEN yn=yc+2
1710 LOCATE xc,yc:CALL &BB8D:t=TIME:WHILE TIME<t+20:W
END
1720 IF xc=xn AND yc=yn THEN RETURN
1730 LOCATE #4,1,2:PRINT#4,SPACE$(5);
1740 xm=1ft+xn-1:IF xm<10 THEN LOCATE #4,2,2:PRINT#4,
MID$(STR$(xm),2) ELSE LOCATE #4,1,2:PRINT #4,MID$(STR
$(xm),2);
1750 ym=top+INT(yn/2)-1:IF ym<10 THEN LOCATE #4,5,2:P
RINT#4,MID$(STR$(ym),2) ELSE LOCATE #4,4,2:PRINT #4,M
ID$(STR$(ym),2);
1760 xc=xn:yc=yn:RETURN
1770 LOCATE #4,xc4,yc4:t=TIME:WHILE TIME<t+20:WEND:PR
INT #4,CHR$(233):IF yc4=3 AND dp=0 THEN 1800:REM CURS
OR 4 ***
1780 IF mo=242 THEN xn4=xn4-1
1790 IF mo=243 THEN xn4=xn4+1
1800 IF mo=240 AND yc4=3 THEN yn4=1
1810 IF mo=241 AND yc4=1 THEN yn4=3
1820 IF yn4=1 THEN xn4=MAX(1,xn4):xn4=MIN(5,xn4):ELSE
xn4=MAX(2,xn4):xn4=MIN(4,xn4)
1830 IF mo=241 AND yn4=3 AND dp=0 THEN xn4=3
1840 LOCATE #4,xc4,yc4:t=TIME:WHILE TIME<t+20:WEND:IF
yc4=1 THEN ON xc4 GOTO 1860,1870,1880,1890,1900
1850 ON xc4 GOTO 1880,1910,1920,1930
1860 PRINT #4,"N":GOTO 1940
1870 PRINT #4,"S":GOTO 1940
1880 PRINT #4," ":GOTO 1940
1890 PRINT #4,"E":GOTO 1940
1900 PRINT #4,"W":GOTO 1940
1910 PRINT #4,"R":GOTO 1940
1920 PRINT #4,"F":GOTO 1940
1930 PRINT #4,"+":
1940 xc4=xn4:yc4=yn4:RETURN
1950 top=1:1ft=1:xc=10:yc=20:xn=10:yn=20:xm=10:ym=10:
REM INITIAL ***
1960 xc4=3:yc4=1:xn4=3:yn4=1
1970 CALL 29003,30997+(top-1)*ncols*6+1ft*3,exs
1980 GOSUB 1200:REM DISPARMY
1990 RETURN
2000 IF xc4=1 THEN top=top-6:top=MAX(top,1):REM NEWMA
P ***
2010 IF xc4=2 THEN top=top+6:top=MIN(top,lwr)
2020 IF xc4=4 THEN 1ft=1ft+10:1ft=MIN(1ft,rgt)
2030 IF xc4=5 THEN 1ft=1ft-10:1ft=MAX(1ft,1)

```

```

2040 CALL 29003,30997+(top-1)*ncols*6+1ft*3,exs:GOSU
B 1200:REM DISPARMY
2050 RETURN
2060 CLS #1:CLS #2:CLS #3:CLS #4:GOSUB 1950:REM INITI
AL:REM PHASE ***
2070 FOR j=1 TO PEEK(30001):FOR k=1 TO maxu:d(j,k)=0:
NEXT:NEXT
2080 PRINT#1,"TURN"INT((turn-1)/gturn+1)m$;
2090 f$=aname$(na)+" army":GOSUB 3960:REM MESSAGE
2100 CLS #3:mo=0:WHILE mo<>32 AND mo<>97 AND NOT(mo=1
00 AND fp=1) AND NOT(mo=109 AND fp=1) AND mo<>114: GO
SUB 1580:WEND:REM COMMAND
2110 IF wind=0 THEN GOTO 2140
2120 IF xc4=3 AND yc4=3 THEN CLS #0:CLS# 1:CLS #2:CLS
#3:CLS #4:RETURN
2130 GOSUB 2000:GOTO 2100:REM NEWMAP
2140 u=1:usq=0
2150 IF mo=109 THEN GOSUB 3990:BORDER 13:GOTO 2100:RE
M MORTAR
2160 uf=usq:uz=u:nz=na:xz=xm:yz=ym:GOSUB 2270:u=uz:x f
=xm:yf=ym:usq=uf:REM FINDUNIT
2170 IF uf=0 THEN PRINT#3,"Roster complete";t=TIME:W
HILE TIME<t+250:WEND:CLS #3:GOTO 2100
2180 xp=xc:yp=yc-1
2190 ny=na:scr=3:vx=1:vy=1:GOSUB 1470:t=TIME:WHILE TI
ME<t+400:WEND:REM UNITLINE/ATTACK
2200 IF mo=114 THEN mo=1000:WHILE NOT(mo=32 AND fp=1)
AND mo<>97 AND NOT(mo=100 AND fp=1) AND mo<>114 AND
mo<>0:GOSUB 4380:WEND
2210 IF mo=0 THEN mo=114
2220 IF mo=114 THEN 2160
2230 IF mo=32 AND fp=1 THEN BORDER 3:GOSUB 3420:REM F
IRE
2240 IF mo=97 THEN BORDER 1:POKE bu+4,PEEK(bu+4) AND
127:GOSUB 1000:REM ADVANCE
2250 IF mo=100 THEN GOSUB 4610:REM DIGIN
2260 BORDER 13:GOTO 2100
2270 IF uf THEN uz=uz+1:bu=aa(nz)+9*uz+2:GOTO 2290:RE
M FINDUNIT ***
2280 bu=aa(nz)+11
2290 IF uz>units(nz) THEN uf=0:RETURN
2300 IF PEEK(bu+1)=yz AND PEEK(bu+2)=xz THEN uf=1:RET
URN
2310 uz=uz+1:bu=bu+9:GOTO 2290
2320 IF u1 THEN CALL 29332:GOTO 2380:REM PROX ***
2330 POKE 29227,units(p1)
2340 bu=aa(p1)+3+9*PEEK(29227):POKE 29221,INT(bu/256)
:POKE 29220,bu-PEEK(29221)*256
2350 POKE 29228,units(p2)
2360 bu=aa(p2)+3+9*PEEK(29228):POKE 29223,INT(bu/256)
:POKE 29222,bu-PEEK(29223)*256
2370 POKE 29226,lim:CALL 29234
2380 u1=PEEK(29227):u2=PEEK(29228):RETURN
2390 nxy=1:u1=0:cvt=0:REM CLOSE ***
2400 vx=5:vy=4:scr=0:lim=1:GOSUB 2320:REM PROX
2410 IF u1=0 THEN RETURN
2420 xm=PEEK(29231):ym=PEEK(29230)
2430 cdone=0:FOR j=1 TO nxy-1:IF xm=xy(1,j) AND ym=xy
(2,j) THEN cdone=1
2440 NEXT:IF cdone THEN 2400
2450 xy(1,nxy)=xm:xy(2,nxy)=ym:nxy=nxy+1
2460 PAPER 0:PEN 1:CLS:PRINT "Close combat at:":PRINT
"Column"x", Row"y
2470 uz=1:uf=0:cn1=0:cn2=0:loss1=0:loss2=0
2480 nz=p1:xz=xm:yz=ym
2490 GOSUB 2270:REM FINDUNIT
2500 IF uf=0 THEN 2530
2510 cn1=cn1+1:com(1,cn1)=uz:com(3,cn1)=PEEK(bu+7):IF
(com(3,cn1)=100) AND (PEEK(bu+4) AND 3=2) THEN com(3

```

```

,cn1)=60
2520 GOTO 2490
2530 uz=1:nz=p2
2540 GOSUB 2270:REM FINDUNIT
2550 IF uf=0 THEN 2580
2560 cn2=cn2+1:com(2,cn2)=uz:com(4,cn2)=PEEK(bu+7):IF
  (com(4,cn2)=100) AND ((PEEK(bu+4) AND 3)=2) THEN com
  (4,cn2)=60
2570 GOTO 2540
2580 PEN 3:PRINT aname$(p1);" Army:"
2590 ny=p1:FOR j=1 TO cn1:u=com(1,j):GOSUB 1470:vy=vy
  +1:NEXT
2600 PEN 3:PAPER 0:PRINT aname$(p2);" Army:";vy=vy+1
2610 ny=p2:FOR j=1 TO cn2:u=com(2,j):GOSUB 1470:vy=vy
  +1:NEXT
2620 ctn1=0:ctn2=0
2630 WHILE ctn1<cn1 OR ctn2<cn2:IF RND(1)<0.5 THEN 26
  60
2640 WHILE ctn1<cn1:ctn1=ctn1+1:cu1=com(1,ctn1):ctnr=
  INT(RND(1)*cn2)+1:cu2=com(2,ctnr):cp1=com(3,ctn1):cp2
  =com(4,ctnr):GOSUB 2860:REM RESCOM
2650 dcp2=(dcp2*(1+FNranorm/50))/cevf:com(4,ctnr)=MAX
  (0,cp2-dcp2):WEND
2660 WHILE ctn2<cn2:ctn2=ctn2+1:cu2=com(2,ctn2):ctnr=
  INT(RND(1)*cn1)+1:cu1=com(1,ctnr):cp2=com(4,ctn2):cp1
  =com(3,ctnr):GOSUB 2860:REM RESCOM
2670 dcp1=(dcp1*(1+FNranorm/50))/cevf:com(3,ctnr)=MAX
  (0,cp1-dcp1):WEND:WEND
2680 FOR j=1 TO cn1:bu=aa(p1)+2+com(1,j)*9:POKE bu+7,
  MAX(0,ROUND(com(3,j))):IF com(3,j)<1 THEN POKE bu+1,0
  :POKE bu+2,0:POKE bu+4,3:loss1=loss1+1
2690 NEXT
2700 FOR j=1 TO cn2:bu=aa(p2)+2+com(2,j)*9:POKE bu+7,
  MAX(0,ROUND(com(4,j))):IF com(4,j)<1 THEN POKE bu+1,0
  :POKE bu+2,0:POKE bu+4,3:loss2=loss2+1
2710 NEXT
2720 LOCATE 1,vy:PAPER 0:PEN 1:PRINT"Combat potential
  ":"PEN 13
2730 PRINT TAB(3);aname$(p1);" = ";
2740 tpot=0:FOR j=1 TO cn1:tpot=com(3,j)+tpot:NEXT:PR
  INT ROUND(tpot)
2750 PRINT TAB(3);aname$(p2);" = ";
2760 tpot=0:FOR j=1 TO cn2:tpot=com(4,j)+tpot:NEXT:PR
  INT ROUND(tpot)
2770 IF loss1>0 OR loss2>0 THEN PEN 1:PRINT"Units eli
  minated:"PEN 11
2780 IF loss1=0 THEN 2810
2790 PRINT TAB(3);aname$(p1);" = ";:FOR j=1 TO cn1:IF
  com(3,j)=0 THEN PRINT CHR$(1);CHR$(PEEK(aa(p1)+5+com
  (1,j)*9));
2800 NEXT:PRINT
2810 IF loss2=0 THEN 2840
2820 PRINT TAB(3);aname$(p2);" = ";:FOR j=1 TO cn2:IF
  com(4,j)=0 THEN PRINT CHR$(1);CHR$(PEEK(aa(p2)+5+com
  (2,j)*9));
2830 NEXT
2840 PEN 2:LOCATE 2,22:PRINT"North to continue";
2850 mo=0:WHILE mo<>240:GOSUB 4380:WEND:GOTO 2400:REM
  MOVE
2860 cev1=PEEK(bcev+cevt*cevr*cevr+cevr*(PEEK(aa(p1)+
  2+9*cu1)-1)+PEEK(aa(p2)+2+9*cu2)-1):REM RESCOM ***
2870 cev2=PEEK(bcev+cevt*cevr*cevr+cevr*(PEEK(aa(p2)+
  2+9*cu2)-1)+PEEK(aa(p1)+2+9*cu1)-1)
2880 dcp1=cev2*cp2:dcp2=cev1*cp1:RETURN
2890 CLS #1:CLS #2:CLS #3:PAPER 0:PEN 1:vx=1:scr=0:R
  EM TURNEND ***
2900 u1=1:ny=na:WHILE u1<=units(na):CLS
2910 PRINT"End of turn"INT((turn-1)/gturn+1):PRINT"fo
  r "aname$(na);" army:"

```

```

2920 vy=4:ny=na:FOR u=u1 TO MIN(units(na),u1+15):GOSU
B 1470:REM UNITLINE
2930 PAPER 0:PEN 1:LOCATE 13,vy:PRINT PEEK(bu+2):LOCA
TE 17,vy:PRINT PEEK(bu+1)
2940 vy=vy+1:NEXT u1:u1=u1+16
2950 PRINT:PEN 2:PRINT TAB(2)"North to continue"
2960 mo=0:WHILE mo<>240:GOSUB 4380:WEND:WEND:REM MOVE

2970 CLS:LOCATE 1,2:PRINT"North for next move":PRINT:
PRINT"SPACE to save game"
2980 mo=0:WHILE mo<>240 AND mo<>32:GOSUB 4380:WEND:RE
M MOVE
2990 CLS:RETURN
3000 PAPER 0:PEN 1:MODE 1:INPUT"File name";file$:REM
SAVE ***
3010 SAVE file$,B,29997,FN deek(29997)-29996
3020 RETURN
3030 MODE 0:INK 14,13:BORDER 13:PAPER 14:CLS:REM STAR
T ***
3040 PRINT "LOADING DATA":LOAD "MCODE"
3050 PRINT:PRINT"New game? (Y/N)"
3060 a$=UPPER$(INKEY$):IF a$<>"Y" AND a$<>"N" THEN 30
60
3070 IF a$="N" THEN PRINT:INPUT"File name";file$:LOAD
UPPER$(file$):GOTO 3090
3080 LOAD "ATTACKD"
3090 DEFINT j-n,u,x,y
3100 DEF FNdeek(add)=PEEK(add)+256*PEEK(add+1)
3110 DEF FNranorm=((RND(1)+RND(1)+RND(1)+RND(1))/4-0.
5)/0.144
3120 GOSUB 4640: REM SYMBOLS
3130 map=31000:nrows=PEEK(30998):ncols=PEEK(30999):cm
ap=37002:amap=37004+PEEK(cmap-1)*PEEK(cmap-2):exs=(nc
ols-20)*3:bcev=amap+2+PEEK(amap-1)*PEEK(amap-2):cevr=
PEEK(bcev-1)
3140 FOR j=1 TO PEEK(30001):aa(j)=FNdeek(30000+j*2):u
nits(j)=PEEK(aa(j)+8):pa(j)=PEEK(aa(j)+9):pe(j)=PEEK(
aa(j)+10)
3150 FOR k=0 TO 7:aname$(j)=aname$(j)+CHR$(PEEK(aa(j)
+k)):NEXT:NEXT
3160 maxu=0:FOR j=1 TO PEEK(30001):maxu=MAX(maxu,unit
s(j)):NEXT
3170 DIM d(PEEK(30001),maxu),t(maxu),com(4,maxu),xy(
2,maxu)
3180 lwr=nrows-10:rgt=ncols-19
3190 WINDOW #0, 1,20,1,22
3200 WINDOW #1, 1,15,23,23:PAPER #1,8:PEN #1,5
3210 WINDOW #2, 1,15,24,24:PAPER #2,14:PEN #2,15
3220 WINDOW #3, 1,15,25,25:PAPER #3,11
3230 WINDOW #4, 16,20,23,25:PAPER #4,9:PEN #4,3
3240 CLS:CLS #0:CLS #1:CLS #2:CLS #3:CLS #4
3250 RETURN
3260 FOR nz=1 TO PEEK(30001):REM PATCH ***
3270 uz=1:uf=0:PAPER pa(nz):PEN pe(nz)
3280 GOSUB 2270:REM FINDUNIT
3290 IF uf=0 THEN 3330
3300 IF (PEEK(bu+4) AND 3)=3 OR (PEEK(bu+4) AND 8) OR
((PEEK(bu+4) AND 4) AND nz<>numarmy) THEN 3280
3310 LOCATE xz-1ft+1,(yz-top)*2+1:PRINT CHR$(PEEK(bu+
3));
3320 IF uz<units(nz) THEN 3280
3330 NEXT:RETURN
3340 IF mf=0 THEN 3380:REM MENDIT ***
3350 sqb=map+(yf-1)*ncols*6+(xf-1)*3
3360 PAPER PEEK(sqb):PEN PEEK(sqb+1)
3370 LOCATE xp,yp:PRINT CHR$(1);CHR$(PEEK(sqb+2));

3380 POKE bu+1,ym:POKE bu+2,xm

```

```

3390 IF mf THEN xz=xf:yz=yf:GOSUB 3260
3400 xz=xm:yz=ym:GOSUB 3260
3410 RETURN
3420 GOSUB 4380:REM FIRE ***
3430 IF mo<>0 THEN GOSUB 1180:REM PIP
3440 IF mo=32 OR mo=103 THEN PRINT CHR$(7);:GOTO 3470

3450 GOSUB 1660:REM CURSOR0
3460 xc=xn:yc=yn:GOTO 3420
3470 IF xf=xm AND yf=ym THEN RETURN
3480 IF d(na,u) THEN f$="Already fired":GOSUB 3960:RE
TURN
3490 pb=PEEK(aa(na)+6+u*9) AND 64:IF pb=64 THEN f$="U
nit panicking":GOSUB 3960:RETURN
3500 pb=PEEK(aa(na)+6+u*9) AND 3:IF pb=3 THEN f$="Uni
t eliminated":GOSUB 3960:RETURN
3510 BORDER 24:dx=xm-xf:dy=ym-yf:ax=ABS(dx):ay=ABS(dy
):IF ax<2 AND ay<2 THEN 3530
3520 IF mo=32 THEN GOSUB 4420:IF fb>1 THEN f$="Out of
sight":GOSUB 3960:RETURN:REM LOS/MESSAGE
3530 bu=aa(na)+7+u*9:pb=PEEK(bu):IF na=1 THEN ne=2:EL
SE ne=1
3540 gr=0:IF mo<>32 THEN gr=pb AND 3
3550 IF mo=103 AND gr=0 THEN f$="No grenades":GOSUB 3
960:RETURN
3560 IF gr>0 THEN POKE bu,(PEEK(bu)AND 240)+gr-1
3570 d(na,u)=1:FOR j=1 TO maxu:t(j)=0:NEXT tt=1:uz=1:
uf=0:nz=ne:xz=xm:yz=ym:GOSUB 2270
3580 IF uf=0 THEN f$="Target empty":GOSUB 3960:RETURN
3590 t(tt)=uz:tt=tt+1:xz=xm:yz=ym:GOSUB 2270:IF uf=1
THEN 3590
3600 tt=tt-1:ut=t(INT(RND(1)*tt+1)):be=aa(ne)+2+ut*9

3610 pe=PEEK(be+4) AND 3
3620 IF pe=3 THEN f$="Target empty":GOSUB 3960:RETURN

3630 ra=INT(SQR(ax*ax+ay*ay))
3640 IF mo=103 THEN sd=0.008:maxr=2:pn=0.5:GOTO 3680

3650 IF pb>127 THEN sd=0.0015:maxr=18:pn=0.3:GOTO 368
0
3660 IF pb>63 THEN sd=0.0025:maxr=4:pn=0.7:GOTO 3680
3670 sd=0.004:maxr=9:pn=0.25
3680 sd=sd*(1+0.1*(5-PEEK(bu+6))):IF (PEEK(bu+4) AND
3) THEN sd=sd*1.5
3690 pn=pn*(1+0.1*(PEEK(bu+6)-5)):IF (PEEK(bu+4) AND
3) THEN pn=pn*0.75
3700 f$="Missed":IF ra>maxr THEN GOSUB 3960:RETURN
3710 IF ra>maxr/2 THEN p=pn*(1-2*ra-maxr)/maxr:r=RND(
1)*pn:IF p>r THEN GOSUB 3960:RETURN
3720 IF ra<=maxr/2 THEN pdh=ABS(FNranorm)*sd*ra:pdv=A
BS(FNranorm)*sd*ra:IF pdh>0.006 OR pdv>0.026 THEN GOS
UB 3960:RETURN
3730 co=PEEK(cmap+xm-1+ncols*(ym-1)):IF (PEEK(be+4) A
ND 32) THEN co=2
3740 rs=RND(1):IF co>0 THEN 3770
3750 IF rs<0.1 THEN 3860
3760 GOTO 3880
3770 IF co=2 THEN 3820
3780 IF rs<0.07 THEN 3860
3790 IF rs<0.6 THEN 3880
3800 IF rs<0.8 THEN 3900
3810 GOTO 3920
3820 IF rs<0.05 THEN 3860
3830 IF rs<0.1 THEN 3880
3840 IF rs<0.2 THEN 3900
3850 GOTO 3920
3860 POKE be+1,0:POKE be+2,0:POKE be+4,3:f$="Target k
illed":GOSUB 3960:mf=1:xf=xm:yf=ym

```

```

3870 xp=xc:yp=yc-1:GOSUB 3340:RETURN
3880 pw=PEEK(be+4) AND 3:IF pw=2 THEN 3860
3890 POKE be+4,(PEEK(be+4)AND 252)+2+pw:f$="Target w
ounded":GOSUB 3960:RETURN
3900 pb=PEEK(be+4) AND 64:IF pb=64 THEN 3880
3910 POKE be+4,PEEK(be+4)+64:f$="Target panicked":GOS
UB 3960:RETURN
3920 f$="Behind cover":GOSUB 3960:RETURN
3930 IF tx=xf AND ty=yf OR tx=xm AND ty=ym THEN RETUR
N:REM BLOCKED ***
3940 t=PEEK(cmap+tx-1+ncols*(ty-1)): IF t>0 THEN fb=f
b+1
3950 RETURN
3960 CLS#2:AFTER 250 GOSUB 3980:REM MESSAGE ***
3970 GOSUB 1190:LOCATE #2,1,1:PRINT#2, f$;SPACE$(15-L
EN(f$)):RETURN
3980 CLS #2:RETURN:REM CLEAR2 ***
3990 BORDER 24:bu=aa(na)+2:mu=0:mc=0:REM MORTAR ***
4000 WHILE mc<mor AND mu<units(na):bu=bu+9:mu=mu+1
4010 IF (PEEK(bu+5) AND 32) THEN mc=mc+1
4020 WEND
4030 IF mc<mor THEN f$="Mortars fired":GOSUB 3960:RET
URN:REM MESSAGE
4040 pbu=PEEK(bu+4) AND 3:IF pbu=3 THEN f$="Eliminate
d":GOSUB 3960:RETURN
4050 IF (PEEK(bu+4) AND 64) THEN f$="Panicked":GOSUB
3960:RETURN
4060 uz=1:uf=0:xz=PEEK(bu+2):yz=PEEK(bu+1):nz=na
4070 GOSUB 2270:REM FINDUNIT
4080 IF uz=mu THEN 4070
4090 IF uf=0 THEN f$="Unassisted":GOSUB 3960:RETURN
4100 rmax=50:xw=xz:yw=yz:sda=0.015:sdb=0.04
4110 IF pbu=2 THEN sda=sda*1.5:sdb=sdb*1.5
4120 GOSUB 4280:REM PROJECTILE
4130 xz=xi:yz=yi:FOR nz=1 TO PEEK(30001):uf=0:uz=1
4140 GOSUB 2270:IF uf=0 THEN 4160:REM FINDUNIT
4150 f$=aname$(nz)+" killed":GOSUB 3960:POKE bu+4,3:m
f=1:xf=xi:yf=yi:xp=xi-1ft+1:yp=(yi-top+1)*2-1:xm=0:ym
=0:GOSUB 3340:GOTO 4140
4160 NEXT
4170 FOR xad=xi-1 TO xi+1:FOR yad=yi-1 TO yi+1
4180 IF xad<1 OR xad>ncols OR yad<1 OR yad>nrows THEN
4260
4190 FOR nz=1 TO PEEK(30001):uf=0:uz=1
4200 xz=xad:yz=yad:GOSUB 2270:IF uf=0 THEN 4250:REM F
INDUNIT
4210 IF RND(1)<0.2 AND PEEK(bu+4)>127 THEN f$=aname$(
nz)+" wnd. ":GOSUB 3960:POKE bu+4,(PEEK(bu+4) AND 25
2) OR 2:GOTO 4200
4220 IF PEEK(bu+4)>127 THEN f$=aname$(nz)+" dug in":G
OSUB 3960:GOTO 4200
4230 IF RND(1)<0.4 OR (PEEK(bu+4) AND 3)=2 THEN f$=an
ame$(nz)+" killed":GOSUB 3960:POKE bu+4,3:mf=1:xf=xad
:yf=yad:xp=xad-1ft+1:yp=(yad-top+1)*2-1:xm=0:ym=0:GOS
UB 3340:GOTO 4200
4240 POKE bu+4,(PEEK(bu+4) AND 252) OR 2:f$=aname$(nz
)+" wnd. ":GOSUB 3960:GOTO 4200
4250 NEXT
4260 NEXT:NEXT
4270 mor=mor+1:RETURN
4280 xdif=xm-xw:ydif=yw-ym:rge=SQR(xdif*xdif+ydif*ydif
):rf=rge/rmax:REM PROJECTILE ***
4290 IF ydif=0 THEN angb=PI-SGN(xdif)*PI/2:GOTO 4330
4300 IF xdif=0 THEN angb=PI/2-SGN(ydif)*PI/2:GOTO 433
0
4310 angb=ATN(xdif/ydif):IF ydif<0 THEN angb=angb+PI
4320 IF angb<0 THEN angb=angb+2*PI
4330 anga=ATN(rf/SQR(-rf*rf+1))/2

```

```

4340 ea=FNranorm*sda:rge=rmax*SIN(2*(ang+ea))
4350 eb=FNranorm*sdb
4360 xi=xw+rge*SIN(angb+eb):yi=yw-rge*COS(angb+eb)
4370 RETURN
4380 j$=LOWER$(INKEY$):REM MOVE ***
4390 IF j$="" THEN mo=0:RETURN
4400 mo=ASC(j$)
4410 RETURN
4420 IF ax<>ay THEN 4480:REM LOS ***
4430 i=0:fb=0
4440 i=i+1:tx=xf+SGN(dx)*i:ty=yf+SGN(dy)*i:GOSUB 3930

4450 IF fb>1 THEN RETURN
4460 IF i<ax-1 THEN 4440
4470 RETURN
4480 IF ax<ay THEN 4550
4490 i=-0.5:fb=0
4500 i=i+1:iy=yf-0.5+i*dy/ax:in=INT(iy):IF in=iy THEN
4530
4510 tx=ABS(i-0.5+SGN(dx)*x):ty=in+1:GOSUB 3930:IF f
b>1 THEN RETURN
4520 tx=tx+1:GOSUB 3930:IF fb>1 THEN RETURN
4530 IF i<ax-0.5 THEN 4500
4540 RETURN
4550 i=-0.5:fb=0
4560 i=i+1:ix=xf-0.5+i*dx/ay:in=INT(ix):IF in=ix THEN
4590
4570 ty=ABS(i-0.5+SGN(dy)*y):tx=in+1:GOSUB 3930:IF f
b>1 THEN RETURN
4580 ty=ty+1:GOSUB 3930:IF fb>1 THEN RETURN
4590 IF i<ay-0.5 THEN 4560
4600 RETURN
4610 IF d(na,u) THEN f$="Already moved":GOSUB 3960:RE
TURN:REM. DIGIN ***
4620 POKE bu+4,PEEK(bu+4) XOR 128
4630 d(na,u)=1:ny=na:scr=3:vx=1:vy=1:GOSUB 1470:t=TIM
E:WHILE TIME<t+400:WEND:CLS #3:RETURN:REM UNITLINE/AT
TACK
4640 CALL 29200,28824,234:REM SYMBOLS/ATTACK ***
4650 SYMBOL 234,48,48,24,28,56,104,44,100
4660 SYMBOL 235,24,24,12,255,14,26,35,65
4670 SYMBOL 236,0,8,4,254,35,33,32,0
4680 SYMBOL 237,0,0,248,28,2,1,0,0
4690 SYMBOL 238,13,77,122,28,120,72,200,4
4700 SYMBOL 239,0,96,48,24,44,71,131,0
4710 SYMBOL 240,12,12,24,56,28,22,52,38
4720 SYMBOL 241,24,24,48,255,112,88,196,130
4730 SYMBOL 242,0,0,127,104,136,8,0,0
4740 SYMBOL 243,176,178,94,56,30,18,19,32
4750 SYMBOL 244,0,6,12,24,52,226,193,0
4760 RETURN

```

This listing is rather a long one, but there is the one big compensation that most of the listing will be used for the other games too. The listing consists of a very short main program (only 20 lines) together with many subroutines (770 lines). Most of the subroutines are standard ones, which are used in all the games programs in the book.

When you have saved "ATTACK", load and save "MCODE" (see Chapter 3). This should be saved on to the same disk as "ATTACK" or immediately after it, on tape. Finally, you need a data file which holds all the information about terrain and armies.

Readers who have typed in the wargaming utility programs should use these to build up the data file, as explained in the next section (Using the utilities).

Keying-in method

Readers who wish to get the game up and running without further ado should proceed as follows.

First type in this *cipher program*.

```

10 REM *** ATTACKC ***
20 MEMORY 28823:CLS
30 PRINT"Storing army data"
40 FOR j=29997 TO 30364
50 READ x:POKE j,x
60 NEXT
70 PRINT"Storing map data"
80 READ x:POKE 30998,x:READ x:POKE 30999,x
90 j=31000:WHILE j<33999
100 READ x$:x=VAL("&"&x$):POKE j,(x AND 240)/16:POKE
j+1,x AND 15
110 READ x$:x$="&"&x$:POKE j+2,VAL(x$)
120 j=j+3:WEND
130 PRINT"Storing cover table"
140 READ x:POKE 37000,x:READ x:POKE 37001,x
150 j=37002:WHILE j<37502:READ n:READ x
160 jj=j+n-1:FOR k=j TO jj:POKE k,x:NEXT
170 j=jj+1:WEND
180 PRINT"Storing movement table"
190 READ x:POKE 37502,x:READ x:POKE 37503,x
200 j=37504:WHILE j<38004:READ n:READ x
210 jj=j+n-1:FOR k=j TO jj:POKE k,x:NEXT
220 j=jj+1:WEND
230 PRINT"Storing CEV table"
240 FOR j=38004 TO 38041
250 READ x:POKE j,x
260 NEXT
270 SAVE "ATTACKD",B,29997,8045
280 PRINT "ATTACKD saved"
290 PRINT "Backup (Y/N)?"
300 a$="":WHILE a$<>"N" AND a$<>"Y"
310 a$=UPPER$(INKEY$):WEND
320 IF a$="Y" THEN 270
330 END

1000 DATA 153,148,0,0,2,58,117,249,117,0,0,0,65,85,83,83
1010 DATA 73,69,32,32,20,3,4,1,0,0,234,0,66,9,100,8,2
1020 DATA 0,0,235,0,2,9,100,8,2,0,0,235,0,2,9,100,8
1030 DATA 2,0,0,235,0,2,9,100,8,2,0,0,235,0,2,9,100
1040 DATA 8,2,0,0,235,0,2,9,100,8,2,0,0,235,0,2,9
1050 DATA 100,8,2,0,0,235,0,2,9,100,8,2,0,0,235,0,2
1060 DATA 9,100,8,3,0,0,236,0,128,9,100,8,4,0,0,237,0
1070 DATA 66,7,100,8,5,0,0,238,0,2,7,100,8,5,0,0,238
1080 DATA 0,2,7,100,8,5,0,0,238,0,2,7,100,8,5,0,0
1090 DATA 238,0,2,7,100,8,5,0,0,238,0,2,7,100,8,5,0
1100 DATA 0,238,0,2,7,100,8,5,0,0,238,0,2,7,100,8,5
1110 DATA 0,0,238,0,2,7,100,8,6,0,0,239,0,32,9,100,8
1120 DATA 74,65,80,65,78,69,83,69,17,0,1,1,0,0,240,128,66
1130 DATA 9,100,9,1,0,0,240,128,66,9,100,9,1,0,0,240,128
1140 DATA 66,9,100,9,2,0,0,241,128,3,7,100,9,2,0,0,241
1150 DATA 128,3,7,100,9,2,0,0,241,128,3,7,100,9,2,0,0
1160 DATA 241,128,3,7,100,9,3,0,0,242,128,128,9,100,9,3,0
1170 DATA 0,242,128,128,9,100,9,3,0,0,242,128,128,9,100,9,3
1180 DATA 0,0,242,128,128,9,100,9,5,0,0,243,128,3,5,100,9
1190 DATA 5,0,0,243,128,3,5,100,9,5,0,0,243,128,3,5,100
1200 DATA 9,5,0,0,243,128,3,5,100,9,6,0,0,244,128,32,8
1210 DATA 100,9,6,0,0,244,128,32,8,100,9
1220 DATA 25,20
1230 DATA 95,5E,99,20,99,20,95,5E,99,20,95,5E,99,20,99,20,95,5E
1240 DATA 91,97,91,9A,91,C4,91,9A,91,C6,91,9C,99,20,99,20,99,20
1250 DATA 99,20,D1,20,99,20,99,20,99,20,99,20,99,20,99,20,99,20

```



```

1260 DATA 99,20,99,20,91,95,99,20,99,20,99,20,99,20,91,95,95,5E
1270 DATA 99,20,95,5E,99,20,DC,20,99,20,95,5E,95,5E,99,20,99,20
1280 DATA 99,20,95,5E,91,96,91,9A,91,99,99,20,99,20,95,5E,99,20
1290 DATA 91,C1,91,9C,99,20,99,20,99,20,DC,20,99,20,99,20,99,20
1300 DATA 99,20,95,5E,99,20,99,20,91,95,99,20,99,20,95,5E,99,20
1310 DATA 99,20,99,20,99,20,91,95,99,20,99,20,95,5E,DC,20,D1,20
1320 DATA D1,20,D1,20,D1,20,D1,20,9D,D7,99,20,91,95,99,20,99,20
1330 DATA 99,20,99,20,99,20,99,20,95,5E,91,C5,99,20,99,20,99,20
1340 DATA D1,20,D1,20,D1,20,D1,20,D1,20,D1,20,9D,D7,91,95
1350 DATA 95,5E,99,20,99,20,99,20,9D,D6,9D,D7,99,20,91,95,95,5E
1360 DATA 99,20,D9,D4,D1,20,D3,8C,D3,8C,D3,8C,D3,8C,D3,8C,D3,84
1370 DATA D1,20,D1,95,D1,20,D1,20,D1,20,D1,20,D1,20,D1,20,D1,20
1380 DATA D1,95,D1,20,D1,20,D1,20,D1,20,D3,83,D3,83,D3,83,D3,83
1390 DATA D3,83,D3,8B,D3,84,D1,95,D1,20,D1,20,D1,20,D1,20,D1,20
1400 DATA D1,20,D1,20,D1,95,D1,20,D1,20,D1,20,D3,8E,D1,20,D1,20
1410 DATA D1,95,D1,20,D1,20,D1,20,D3,8B,D3,8C,D3,8C,D3,8C,D3,8C
1420 DATA D3,8C,D3,8C,D3,8C,D3,8C,D3,8C,D3,8C,D3,8E,D3,81
1430 DATA D1,20,D1,20,D1,95,D1,20,D1,20,D1,20,D1,20,D3,83,D3,83
1440 DATA D3,83,D3,83,D3,83,D3,83,D3,83,D3,83,D3,83,D3,83,D3,83
1450 DATA D3,81,D1,20,99,20,91,96,91,99,99,20,9D,D5,D1,20,D1,20
1460 DATA D1,95,D1,20,D1,20,D1,20,D1,20,D1,20,D1,20,D1,20,D1,95
1470 DATA D1,20,D1,20,D1,20,9D,D4,95,5E,91,C7,99,20,99,20,95,5E
1480 DATA 9D,D5,D1,20,D1,C7,D1,20,D1,20,D1,20,D1,20,D1,20,D1,20
1490 DATA D1,20,D1,95,D1,20,D1,20,9D,D4,99,20,99,20,91,95,95,5E
1500 DATA 99,20,99,20,99,20,99,20,91,95,99,20,99,20,99,20,9D,D5
1510 DATA D1,20,D1,20,9D,D4,91,95,99,20,99,20,99,20,95,5E,99,20
1520 DATA 91,95,99,20,99,20,99,20,95,5E,99,20,91,95,99,20,95,5E
1530 DATA 95,5E,99,20,99,20,99,20,99,20,91,95,99,20,95,5E,99,20
1540 DATA 99,20,C1,96,C1,99,C1,20,C1,20,9C,D7,99,20,99,20,91,C1
1550 DATA 91,9A,91,C6,91,9C,99,20,95,5E,99,20,95,5E,91,95,99,20
1560 DATA 99,20,99,20,99,20,C1,C5,C1,20,C1,20,C1,20,C1,20,9C,D7
1570 DATA 99,20,95,5E,99,20,99,20,91,95,99,20,99,20,99,20,99,20
1580 DATA 91,C5,95,5E,99,20,99,20,95,5E,C1,C7,C1,20,C1,20,C1,20
1590 DATA C1,20,C1,20,99,20,99,20,99,20,99,20,91,C5,95,5E,99,20
1600 DATA 91,96,91,9A,91,C0,99,20,99,20,99,20,99,20,99,20,C1,C5,C1,20
1610 DATA C1,20,C1,20,C1,20,C1,20,99,20,99,20,99,20,95,5E,91,95
1620 DATA 99,20,99,20,91,95,99,20,99,20,99,20,99,20,95,5E,99,20
1630 DATA C1,95,C1,20,D1,20,D1,20,C1,20,C1,20,9C,D7,95,5E,99,20
1640 DATA 99,20,91,93,91,9A,91,C4,91,9D,99,20,95,5E,99,20,99,20
1650 DATA 99,20,99,20,C1,95,C1,20,D1,20,D1,20,C1,20,C1,20,C1,20
1660 DATA 9C,84,99,20,99,20,99,20,99,20,95,5E,91,95,99,20,99,20
1670 DATA 99,20,99,20,99,20,C1,93,D1,20,D1,20,D1,20,C1,20
1680 DATA C1,20,C1,20,C1,20,95,5E,99,20,99,20,99,20,91,C1
1690 DATA 91,9A,91,9C,99,20,92,D6,2D,20,20,C1,20,D1,20,D1,20
1700 DATA D1,20,C1,20,C1,20,C1,20,C1,20,9C,D7,99,20,95,5E,99,20
1710 DATA 99,20,95,5E,99,20,91,95,95,5E,2D,20,2D,20,2D,20,C1,20
1720 DATA D1,20,D1,20,D1,20,D1,20,C1,20,C1,20,C1,20,C1,20,9C,85
1730 DATA 99,20,99,20,99,20,92,8E,2C,20,21,C7,20,20,2D,20,2D,20
1740 DATA 2D,20,C1,20,D1,20,D1,20,D1,20,D1,20,C1,20,C1,20,C1,20
1750 DATA C1,20,C1,20,99,20,99,20,92,D6,2C,20,2C,20,21,C5,2C,20
1760 DATA 2D,20,2D,20,2D,20,C1,20,D1,20,D1,20,D1,20,D1,20,C1,9A
1770 DATA C1,9C,C1,20,C1,20,9C,D4,92,D6,2C,20,2C,20,2C,20,2D,20
1780 DATA 21,95,2D,20,2D,20,2D,20,C1,20,D1,20,D1,20,D1,20
1790 DATA D1,20,C1,20,C1,C5,C1,20,C1,20,99,20,2C,20,2D,20,2C,20
1800 DATA 2C,20,2C,20,21,95,2C,20,2C,20,2C,20,C1,20,C1,20
1810 DATA D1,20,C1,20,C1,20,C1,20,C1,C7,C1,20,C1,20,99,20,2C,20
1820 DATA 2C,20,2C,20,21,96,21,9A,21,9B,21,9C,2D,20,2C,20,2C,20
1830 DATA C1,20,C1,20,D1,20,C1,20,C1,20,C1,20,C1,C5,C1,20,C1,20
1840 DATA 99,20,2C,20,2C,20,2D,20,21,95,2C,20,2C,20,21,95,2C,20
1850 DATA 2C,20,2C,20,C1,20,C1,20,C1,20,C1,20,C1,20,C1,96,C1,C0
1860 DATA C1,20,9C,D4,99,20,2D,20,2C,20,2C,20,21,95,2B,20,2C,20
1870 DATA 21,C1,21,9C,2C,20,2B,20,C1,20,C1,20,C1,20,C1,20,C1,20
1880 DATA C1,95,C1,20,C1,20,99,20,99,20,2C,20,2C,20,2C,20,21,95
1890 DATA 2C,20,2C,20,2C,20,21,95,2B,20,2C,20,9C,D5,C1,20,C1,20
1900 DATA C1,20,C1,20,C1,95,C1,20,C1,20,95,5E,92,D6,2C,20,2C,20
1910 DATA 21,96,21,99,2C,20,2D,20,2C,20,21,C7,2C,20,2C,20,99,20
1920 DATA 9C,D5,C1,20,9C,85,C1,20,C1,95,C1,20,9C,D4,99,20,2C,20
1930 DATA 2C,20,2D,20,21,95,2C,20,2C,20,2C,20,2C,20,21,C5,2C,20
1940 DATA 2B,20,95,5E,99,20,99,20,99,20,95,5E,91,97,91,C6,91,C4

```

To create this file, all you have to do is to run **ATTACKC**. **ATTACKC** automatically creates the data file **ATTACKD** on the same disk as itself and

the other programs (ATTACK and MCODE). ATTACKD is the data file which is loaded by the game program, ATTACK, when it runs.

With tapes The procedure is slightly different. Save ATTACKC on a separate tape. Now load ATTACKC. Then place the tape which has ATTACK and MCODE already on it, in the recorder, wound on to just beyond the end of the recording of MCODE. Now run ATTACKC. Press 'RECORD' and 'PLAY' and any keyboard key when asked to do so. The data file, ATTACKD, is then saved on the tape, immediately after MCODE. ATTACKD is the data file which is loaded by the game program, ATTACK, when it runs.

After you have used ATTACKC to create ATTACKD, you may never need to use it again. However, it is best to keep the recording in case you have made mistakes in entering the data. It may then be necessary to edit ATTACKC.

With ATTACK, MCODE and ATTACKD on the tape (in that order), or on the same disk, you are ready to play. Type RUN "ATTACK" and press ENTER. ATTACK is loaded and run; it then loads MCODE and ATTACKD, after which the game begins.

Using the utilities

Entering data by means of the utilities is in general easier and less subject to mistakes than simply keying lines of data statements. In addition, it gives you the chance to vary the data to adapt the game to your own requirements. For example, you may like to alter certain features on the map such as the tracks, alter the colours in which the terrain is displayed, or even extend the map to a greater area. You may prefer to omit the tracks altogether, except perhaps where they enter the jungle. In real life, soldiers are unlikely to have maps of minor jungle tracks. So a 'track less' jungle area is more realistic. In the game they will be able to 'find their way' along a track by discovering which direction gives them the greatest movement distance. But they will not know to where the track is leading. You may also wish to alter the composition of the armies so as to obtain a different balance between the forces. Numbers of units, their weapons, and skills can all be altered to fit your own ideas.

For detailed instructions in using the utilities, see the earlier chapters concerned. Below we set out a suggested plan for putting together the data file for ATTACK.

1 RECRUITER. Prepare a file for each army, named 'Aussie' and 'Japanese' respectively. The data required is in Tables 4 and 5. Note that there are six types of unit in *both* armies, but there are none of type 4 in the Japanese army. Starting positions are not specified and there are five details. The values in the 'Arms' column consist of the number of grenades carried, plus a value depending on the type of weapon:

Weapon	Value
Light machine-gun	128
Sub-machine-gun	64
Mortar	32
Rifle	0

Thus, the 'Arms' value for the sergeant is 66, indicating a sub-machine-gun and two grenades.

Table 4 Australian Army

Paper no. 3: Pen no.4: 6 types of unit.

<i>Unit nos.</i>	<i>Type name</i>	<i>Type no.</i>	<i>Symbol</i>	<i>Status</i>	<i>Arms</i>	<i>Weapon skill</i>	<i>Close combat potential</i>	<i>Morale</i>
1	Sergeant	1	234	0	66	9	100	8
2-9	Private/1	2	235	0	2	9	100	8
10	Private/2	3	236	0	128	9	100	8
11	Corporal	4	237	0	66	7	100	8
12-19	Private/3	5	238	0	2	7	100	8
20	Private/4	6	239	0	32	9	100	8

Table 5 Japanese Army

Paper no. 0: Pen no.1: 6 types of unit.

<i>Unit nos.</i>	<i>Type name</i>	<i>Type no.</i>	<i>Symbol</i>	<i>Status</i>	<i>Arms</i>	<i>Weapon skill</i>	<i>Close combat potential</i>	<i>Morale</i>
1-3	Sergeant	1	240	128	66	9	100	9
4-7	Private/1	2	241	128	3	7	100	9
8-11	Private/2	3	242	128	128	9	100	9
12-15	Private/3	5	243	128	3	5	100	9
16-17	Private/4	6	244	128	32	8	100	9

The '128' for status indicates that all Japanese units are already dug in at the start of the game.

2 CARTOGRAPHER. The terrain map has 20 columns and 25 rows. The colours in which the various types of terrain are mapped have been indicated on page 24, but you can choose any other colours you prefer. Figure 2.1 shows the nature of the terrain in each square. Figure 3.5 illustrates the way each square can be mapped, though you are free to map it in any way you please. Save the finished map under the name "JUNGLE".

3 TABLER. Three tables are needed: JUNGLE C for holding cover values for each square of the map; JUNGLE M, for holding movement costs; CEV/ATTACK for the CEV values in close combat. In this game JUNGLE C holds only zeros or ones as there is no level 2 cover. The table consist of 20 columns and 25 rows, as does the map. The table begins at address 37000, the standard base address for a cover table. Using Figure 2.1 as a guide,

enter '1' for squares in which any part of the area is jungle or rocky outcrop, and for the outermost swamp squares. Enter '0' for all other squares.

JUNGLEM also has 20 columns and 25 rows. It begins at address 37502. Using Figure 2.1, key 1 for any square that has road on it and 2 for any square that has a path. The values for other squares depend on terrain occupying the greater area of the square: 2 for open ground, 3 for beach, 4 for rocky outcrop, swamp or jungle, 9 for sea. The value 9 prevents any unit from moving on to a sea square, as 9 is more than the maximum movement allowance.

For the CEV table, which has six rows and six columns and begins at 38004, use the values given in Table 6.

INTELLIGENCE. Use this program to load and store the following files:

- Army 1 AUSSIE
- Army 2 JAPANESE
- Map JUNGLE
- Table 1 JUNGLEC
- Table 2 JUNGLEM
- Table 3 CEV/ATTACK

The complete data file is saved under the name ATTACKD. With disks, it should be on the same disk as the ATTACK and MCODE files. With tape, it should follow after ATTACK and MCODE.

Table 6 CEV/ATTACK

Attacking unit type	Attacked unit type					
	1	2	3	4	5	6
1	54	60	60	60	60	60
2	34	37	37	37	37	37
3	37	40	40	40	40	40
4	37	40	40	40	40	40
5	44	44	48	44	40	54
6	32	34	34	34	34	34

Notes for programmers

The sequence of the main program is:

- 20 Set screen colours.
- 30 Reserve memory; call START to initialise the game.
- 40 Omit deployment phase, if game is resumed.
- 50 Deploy Australian army (first 10 units only).
- 60 Deploy Japanese army.
- 70 Calculate turn number.
- 80 Set units (1) to current size of Australian army.
- 90 Deploy Australian army (second 10 units) in turn 8.

- 100 Determine number of army to play this turn.
- 110 Call PHASE for Advance Phase.
- 120 Call PHASE for Advance/Fire phase.
- 130 Call CLOSE for Close Combat Phase, if required.
- 140 Call BLOCK machine-code routine to rally panicked units.
- 150 Call TURNEND; Save or continue?
- 160 To end game after move 20.
- 170 Back to line 70, if continuing.
- 180 Save game.
- 190–200 Restore mode 1 and its colours. End program.

Subroutines

The following subroutines appear for the first time in this program. Variables and arrays are defined in Appendix A. The way in which the subroutines are related to the main program and to each other is shown in Figure 6.5.

ADVANCE Advances a unit, calculating its movement allowance according to terrain and whether the unit is wounded. Does not allow units to move if panicked, eliminated or already moved.

PIP Generates a brief ‘pip’ sound for registering a key-press.

BOOM Generates a long, low-pitched sound for calling attention to a displayed message.

DISPARMY Displays army symbols in their present locations on the terrain map. Units are not displayed if in reserve, if eliminated, if hidden from the player whose turn it is to play, or if engaged in independent movement.

DEPLOY Controls deployment phase for one army.

UNITLINE/ATTACK Displays unit symbol and details at the bottom of the screen. This subroutine is suitably modified for each game.

COMMAND Accepts commands from the players.

CURSOR0 Controls the flashing and motion of the cursor while it is in window 0 (the map).

CURSOR4 Controls the flashing and motion of the cursor while it is in window 4 (the bottom right corner).

INITIAL Initialises a phase by displaying the top left area of the map, and displaying the army units situated on it.

NEWMAP Displays a new section of the map and the army units situated on it.

PHASE Controls the playing of a phase of the game.

FINDUNIT Scans the data of an army to find which units are located on a given square.

PROX Finds pairs of units of any two armies that are within a given distance of one another.

CLOSE Controls the close combat phase.

RESCOM Resolves combat between two units (see Chapter 4).

TURNEND Displays a roster of the army whose turn has just ended. Gives the option of continuing with the game or saving its present state to tape or disk.

SAVE Saves the present state of the game to tape or disk.
START Initialises a game by loading MCODE and the data file, defining integers, functions and graphics symbols, calculating values of important variables, dimensioning arrays, and defining windows and their colours.

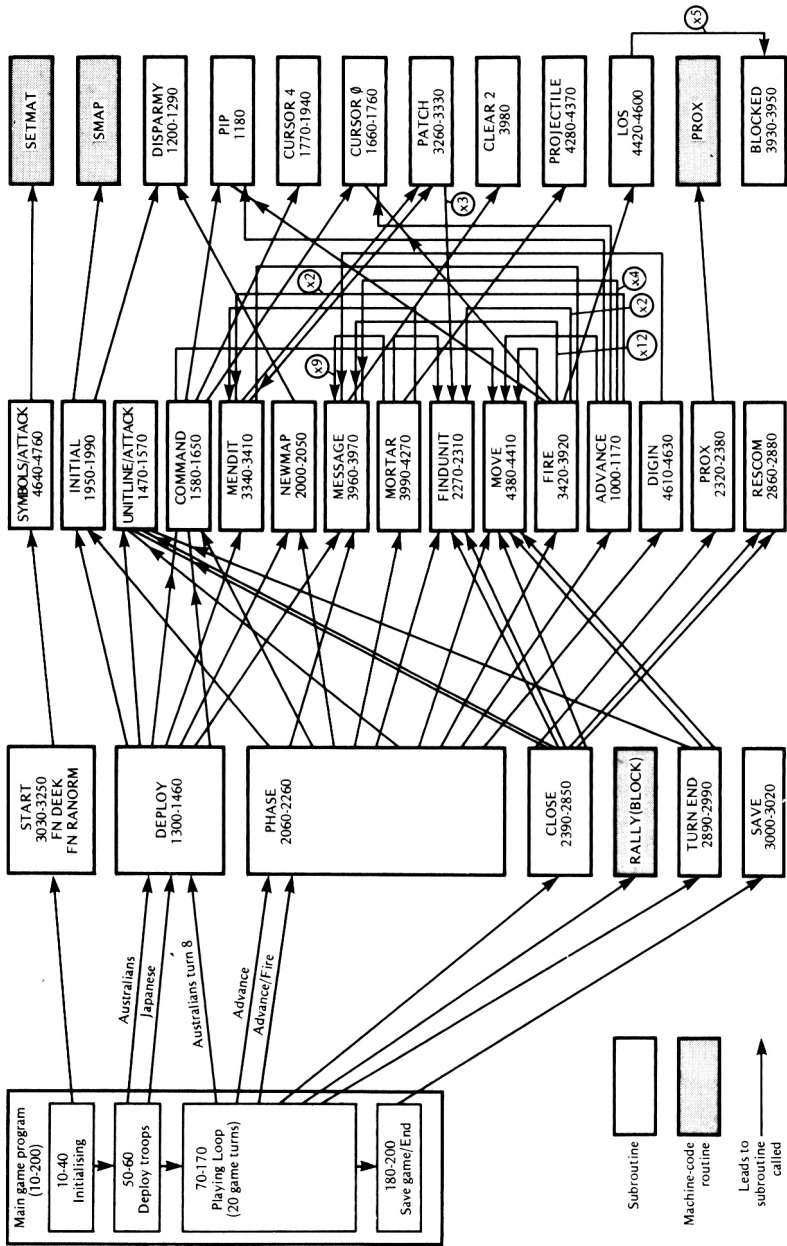


Fig 6.5 How the program and subroutines of JUNGLE ATTACK are related.

PATCH Displays the symbols of the units present on any given square of the map.

MENDIT Moves a unit symbol from one square to another, renewing the map, and re-displaying the symbols of other units present on the squares concerned.

FIRE Calculates the effect of firing weapons such as rifles, revolvers and machine-guns and the throwing of grenades (see Chapter 4). May need slight amendment in different games.

BLOCKED Determines if the line of sight is blocked by a 'cover' square.

MESSAGE Displays a flashing message on the second line up (window 2), for five seconds.

CLEAR2 Clears window 2.

MORTAR Calculates the effect of firing a mortar.

PROJECTILE Calculates the landing point of a projectile such as a mortar bomb (see Chapter 4).

MOVE Reads the keyboard.

LOS Determines whether there is an obstacle blocking the line of sight between the centres of any two squares.

DIGIN Enables a unit to become dug in, or the reverse.

SYMBOLS/ATTACK Defines the unit symbols. This subroutine has different definitions for each game.

7

Hidden movement

The canopy of a tropical jungle is exceedingly dense. From an aircraft flying over such an area, it is almost impossible to observe army units on the ground. Yet, in the JUNGLE ATTACK game, the Japanese player is able to observe the Australians as they advance through the jungle and climb up the rocky slopes. The Australians are able to see Japanese scouting parties moving toward them along the tracks and have ample time to take evasive action. In a real action, opponents creeping through the jungle would come across each other almost without warning and be engaged instantly in hand-to-hand fighting. Clearly, the game lacks realism in these regards.

Surprise is often a decisive tactical advantage and we must introduce this element into our games, if possible. If the players are able to 'hover' over the battle area and know the exact location of both friendly and enemy units, this element is completely lacking from the game. Miniature wargamers and board wargamers face the same problem, the need for *hidden movement*. One way of doing this is by letting the preliminary moves of the game take place off the table. Figures are placed on the table when the players (or umpire, if there is one) decide that members of opposing forces would be visible to one another. Another technique is to hang a curtain across the room, resting across the table. Opponents deploy their forces on opposite sides of the table. The curtain is removed when the moment for the battle arrives. Such a procedure is unsuitable for many scenarios, including that of JUNGLE ATTACK, where there is no clear no-man's-land between the opponents. In board wargames hidden movement is simulated by turning counters upside down to conceal their identity. Sometimes, use is made of dummy counters to confuse the enemy.

The computer simulates hidden movement easily and very effectively. At all stages of the game the computer has a record of the exact location of every unit. It takes less than a second to run through these records and decide which units of each army are visible to the opponent. In a given player's turn, the map shows the player the location of all the units the player is commanding, but only those enemy units which would be visible to units on the player's side. Naturally, in a game which has hidden

movement, players do not look at the screen when it is their opponent's turn to play!

Since it is so adept at simulating hidden movement, the computer can be a valuable adjunct to the miniature wargaming table. The game is conducted on the computer, to keep an updated record of the locations of all units, both visible and invisible. When a unit is visible to both sides a model representing it is place on the table. If it subsequently goes into hiding, the model is removed. Hidden movement was not included in the listing of JUNGLE ATTACK so that this introductory wargame was kept fairly simple. However, hidden movement adds so much to a wargame, whether it is played with models or not, that the reader is recommended to amend JUNGLE ATTACK as described below.

Hidecover

This is a subroutine that makes units hidden on the basis of whether they are in cover or not. To see this in action, load JUNGLE ATTACK and type in the following subroutine:

```
4770 bu=aa(na)+11:FOR j=1 TO units(na):REM HIDECOVER
***
4780 IF PEEK(cmap+ncols*(PEEK(bu+1)-1)+PEEK(bu+2)-1)
OR (PEEK(bu+4)>127) THEN POKE bu+4,PEEK(bu+4) OR 4:EL
SE POKE bu+4,PEEK(bu+4) AND 251
4790 bu=bu+9:NEXT:RETURN
```

Also add these lines to the main program to call the routine after each army has deployed and at the end of each turn:

```
55 GOSUB 4770:REM HIDECOVER

65 GOSUB 4770:REM HIDECOVER

135 GOSUB 4770:REM HIDECOVER
```

When the game is played, all units that are on jungle squares, rocky outcrop squares and the squares at the edge of the swamp, are hidden. They are shown on the map in their own army's turn, but not in the enemy's turn.

In addition to the above, a unit is hidden if it is dug in, even if it is on open ground or on the beach. If you prefer not to incorporate this feature, omit 'OR (PEEK(bu+4)>127)' from line 4780. Also, this should be omitted in any game in which digging in does not occur.

Firing at hidden units

The firing and mortar routines operate as normal with hidden units. The only difference is that you are not able to see the symbol of the enemy unit on the screen. In your Advance/Fire Phase, move the cursor to the square on which you *think* an enemy unit is situated and press the space bar. If no enemy unit is there, the message 'Target empty' appears. If the enemy is present, one of the other possible messages, such as 'Missed', or 'Wounded' appears. This will give you an indication of their hidden positions. If your troops enter a square on which an enemy unit is hidden, you will automatically be in close combat with them. So, advance with caution!

Other hidden movement

Cover is not the only factor that makes a unit invisible to the enemy. Smoke, fog and mist affect visibility. The effect of fog and mist, depend to a large extent on the distance between the opposing units. Another possible factor is the time of day. Routines can be written to take account of these factors.

Obstacles in the line of sight are yet another factor. A unit may be invisible if in the open, but hidden *behind* a building or wooded area. The relief of the terrain is another example of an obstacle in the line of sight. For example, the undulating terrain on which the battle of Naseby was fought provided concealment for large numbers of men. In the NASEBY game program we use a hidden movement subroutine based on line of sight, to cause troops to disappear from sight or to reappear, according to the lie of the land. A fully accurate line of sight routine for hidden movement would involve having a detailed 'relief map' of the area stored in the computer's memory. Assessing the visibility between one point and another would require elaborate trigonometrical calculations, which would take a long time to compute. We have therefore implemented a much simpler routine which has more-or-less the same effect. It also has the advantage of being applicable to blocking the line of sight by obstacles of *any* kind.

HIDE Obst detects whether any square between one unit and any given unit of the enemy, is blocked by an obstacle. It uses the same line of sight subroutine (LOS) that is used by the firing subroutine (FIRE) to decide if a firing unit has a clear view of its target. Obstacles are detected by reference to the cover table. Thus a woodland, jungle, rocky outcrop or building square blocks the line of sight. In NASEBY, we consider that woodland, hedge squares, Naseby town squares and the squares on which the terrain is sloping, count as obstacle squares. The terrain is divided into areas of low ground and high ground. The single line of squares between these areas are sloping squares. The effect of this is shown in Figure 7.1. Units which are both on low ground can see each other, unless there is a high ground between them.

The same applies to two units which are both on high ground, provided that there is no sloping square between them. With the LOS subroutine, a

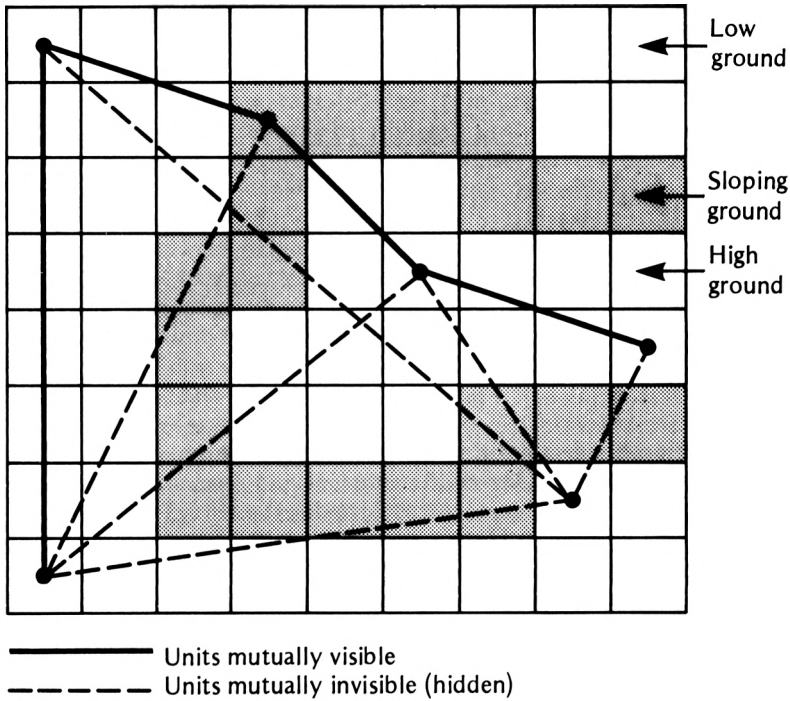


Fig 7.1 Hidden movement, as obtained by subroutine HIDEOBST.

line of sight exists into and out of (but not *through*) an obstacle square. Therefore units that are on a sloping square can see and be seen by units on low or high ground on either side. Obviously the best viewpoint is on the sloping ground (equivalent to being on the edge of a ridge or hill in the real terrain) but there, silhouetted against the skyline, the unit is plainly visible to all around.

8

Briefing

This is the name of a subroutine designed to give the maximum flexibility to the game program. One of the unusual and fascinating features of wargaming is that the players are free to modify the rules. They must agree, of course on any such modifications. On the other hand, although it is desirable to incorporate as many rules as are required into a wargame program, it is not easy to make changes in the program unless one has some experience of BASIC programming. Even with this experience, there can be difficulties if alterations have unforeseen consequences on another part of the program. The more the program takes over the operation of the rules, the less easy it is for the player to modify them.

Let us take an example of this, relating to movement of units. As we shall see later, similar considerations apply to rules covering weapons, combat potential, morale and other aspects of the game. In JUNGLE ATTACK, the computer automatically operates the rules that govern how far a unit may move each turn. It calculates how many movement points are used as a unit moves from square to square. If you attempt to move a unit too far, the 'Too far' message appears and the movement is not allowed. This system is a convenience for the players, and so helps to speed up the play of the game. We could just have easily written an "advancing subroutine" which allows units to be moved any distance and anywhere. This would allow the players to decide on their own movement rules, but they would then have the responsibility of keeping to them. They would also spend a relatively high proportion of playing time in working out movement points costs.

Nevertheless, there may be occasions when players wish to amend the movement rules and let the units move further. For example, in JUNGLE ATTACK they might consider that a unit could, for a limited period, run at the double along the road and jungle paths. They might choose to allow it, say, 10 movement points per turn instead of six. This is not possible with the program as it is written. The players might also decide that the Australian units should be allowed to wade or swim along the coast, so as to scale the south western face of the rocky outcrop. This is not allowed by the rules built into the program. Units cannot move on to a sea square (though, as you may have discovered, there is nothing in the program to prevent you deploying a unit on the sea at the beginning of the game!). As

well as the need to play to a set of rules amended before the game begins, there may also be occasions during a game where special rules have to be agreed to. No simple or even moderately complicated set of rules can cover every possible situation that may arise in the game. This is where wargaming differs so markedly from chess and *Go*. When a situation arises to which the rules provide no guidance, the players decide between themselves on a reasonable course of action. It cannot be guaranteed, however, that the program will allow them to take this course! Suppose, for example that a single Japanese rifleman is advancing along a jungle path. A party of four Australians armed with sub-machine-guns comes toward him. If the Australian player moves his units on to the same square as the Japanese unit, this automatically results in close combat, in which the Japanese soldier would suffer badly. Some players might agree that this correctly models what would happen in a real engagement. Yet the side of a square represents 33 m, while a path may be only 1 m wide. Some players might consider that the Japanese would hear the Australians coming, and have ample time to hide in the vegetation beside the path. He could hide there until they have passed. These players would consider that for the Japanese to be automatically engaged in close combat is unfair. To avoid close combat the Australian units would have to stop short of the square on which the Japanese is located and so lose part of their movement allowance. This too is unfair. If you prefer a clear-cut approach to the game, you may decide that such unfairnesses as these will 'average out' over the course of a game. You will play to a simple set of rules and let the computer program take care of most of these.

But many wargamers take particular delight in putting as much realism as possible into their game. They may decide that the Australians should stop short of the square on which the Japanese is 'hiding'. They could use the BRIEFING routine to move them an extra square next turn, by which time the Japanese will have moved away. If the players are unable to decide whether the Japanese would hear the Australians or not, they could probably use a dice in deciding which course to take in this situation. The players would throw a dice, having previously agreed that, say, with a throw of one or two the Australians move on to the square and attack the Japanese in close combat, but with a throw of three to six they stop short and move an extra square next turn. At this stage in the discussion, one of the players might suggest that if the Japanese unit was already wounded, he would not be able to evade the Australians. Here is another possibility to consider, that is not likely to be covered by the rules. The situation is best judged on its merits at the time it arises. It is advantageous if we work out our own solutions and let the program carry them into effect.

What BRIEFING does

BRIEFING allows a player to change any of the details of any unit under the player's command at any time during the player's turn. The details concerned, and possible reasons for wanting to change them are listed as follows.

1 *Unit type* A good example of this appears in NASEBY. The dragoons are mounted infantry, fighting on the Parliamentary side. They are able to move rapidly about the battlefield on horseback, but may also dismount and fire their muskets as ordinary infantrymen. When they are mounted, we consider them as type 6, mounted dragoon. They move and fight in a similar way to the cavalry. When we wish them to dismount, we change their type number to 4, musketeer, and they move and fight as ordinary infantry. A similar situation could arise in a game set in the 20th century in which infantry units might alternate between being on foot or being transported in vehicles at different stages of the game. The two types would have different movement allowances and would gain protection against fire when inside the vehicle. BRIEFING would be used, in effect, to embark or disembark the units.

2 *Row* Changing row enables a player to over-ride the built-in movement rules. Possible reasons for wishing to do this have been described above.

3 *Column* The same applies as for detail 2.

4 *Symbol* The symbol used to represent the unit on the map can be changed by altering this detail. It might, for example be useful to have a special symbol to indicate a unit taken prisoner. There could be different symbols according to which way a unit is facing. In other games the rules may allow units to be promoted in the field and so require a change of symbol.

5 *Status* The status byte holds a variety of information about the unit. In a skirmish game, we can alter the byte to make an active unit wounded, or make a wounded unit recover. We can reveal a hidden unit or hide one that is not already hidden. In battle games, such as NASEBY and OMDURMAN, we use BRIEFING to change the direction in which the unit is facing. Changing the status detail is complicated by the fact that the data is stored in individual bits of the byte, but special instructions are given with each game.

6 *Arms* In JUNGLE ATTACK, this detail stores the number of grenades carried, and also indicates if the unit is carrying a sub-machine-gun or light machine-gun. Changing this detail enables us to issue a unit with a fresh supply of grenades, for example. Or a unit could take grenades from a captured or killed enemy. Altering this detail also allows weapons to be transferred from one unit to another, or to be captured from an enemy unit.

7 *Weapons skill* Although weapon skill is a factor that is not likely to vary during the course of a game, there may be games in which players may wish to change it. For example, if the game allows that weapons may be transferred from one unit to another (as described above) the unit receiving the weapon may not have been trained to use it. The weapon skill of that unit should be reduced.

8 *Combat potential* In skirmish games, this usually has the value 100 at the start of the game and is diminished automatically as a result of close combat. Players may wish to adjust combat potential up or down to take account of other factors. For example, it might be agreed that if a unit rests for a specified number of turns following close combat, its combat potential may be increased by a given amount. In battle games, the combat potential

relates to the number of warriors in the unit. In some games this number may be equal to the number of men but in other games a scale factor is applied. This is the case in NASEBY where many units contain 750 men. This is too large a number to be stored in a single byte, since the maximum storable value is 255. It would be possible to allocate two detail bytes for the combat potential, but this means that it is necessary to have two kinds of combat resolution routine, depending on whether CP is one byte or two. Also allotting two bytes to CP may leave insufficient bytes for other purposes. Accordingly, in NASEBY the CP value is one third of the number of men in the unit. A unit starting with 750 men has CP of 250 and the scale factor is taken into account when the computer resolves combat. The routine which displays the details of the unit can also take this into account, multiplying the CP by three and displaying the actual number of men.

In battle games there may be occasions when the CP needs to be adjusted by the players. For example, the combat resolution routines do not take account of whether a unit is attacked from the front, the flanks or the rear. The routine assumes that the attack is met to the front. Players may wish to reduce the CP of units attacked from the flanks or the rear. They may also wish to take account of disorganization. Suitable rules to cover these factors are given with the game. These rules may be modified according to the preferences of the players without the need to alter the computer program.

Another situation in which CP may need to be altered is when, if the rules of the game allow it, two or more units are re-formed as a single unit. Units which have suffered heavy casualties to the point of their being likely to be eliminated or to surrender, may be brought together on the same square and one of them allotted the total CP of the group. The CP of the others is reduced to zero and they are eliminated.

9 Morale Morale is governed by so many factors, some of them not easily quantified. It is difficult to write a computer program to take account of them all. All troops may be said to have a given morale level at the beginning of the game but, from then on, this is subject to change. Reasons which cause it to fall include: the leader is killed; the unit suffers a very high casualty rate in one turn; the unit is attacked by a much more powerful enemy unit; a routed unit of its own side passes close by. Reasons which may cause it to rise include; a leader visits the unit; an enemy leader is killed on a nearby square; the unit is attacking an enemy unit from higher ground. If the morale falls below a given level, the unit may surrender or desert.

Operating BRIEFING

In NASEBY and OMDURMAN, the briefing routine can be entered at any phase of the game. If there is one unit on the square, place the cursor under the symbol of the unit you wish to brief and press key 'B'. The current details of the unit are displayed briefly in the normal way. If there is more than one unit on the square, move the cursor to the square, press 'R' to

roster the units, then press 'B' when the details of the unit you require are displayed. The border of the screen then changes to a lilac colour to indicate that you are in briefing mode. The lower line of the screen displays the query 'No.?' on a black background. The computer is asking you which detail number you wish to amend. The details are numbered from one to nine, as in the list above. For the benefit of programmers, these correspond to the bytes *bu* to *bu+8*, as tabulated in Chapter 5.

When you have selected the detail, and pressed ENTER, the bottom line displays the present value of the detail. An arrow pointing toward this, followed by a question mark invite you to indicate the change required. To spare you the unnecessary labour of performing mental calculations when there is a computer available, there are several formats for indicating the changes you wish to make. Key in an arithmetic operator immediately followed by a value (i.e. no space between). The value can be between 0 and 255, but not outside these limits. These are the 5 possible formats:

<i>Format</i>	<i>Example</i>	<i>Effect on present value</i>
=value	=5	Replaced by 5
+value	+5	Increased by 5
-value	-5	Decreased by 5
*value	*5	Multiplied by 5
/value	/5	Divided by 5

Note that the value need not be an integer. For example, if you want to reduce the CP of a unit to 70% of its previous value, the format is '*0.7'.

The results of all calculations are rounded to the nearest whole number. If the result of a calculation is a negative value, the value stored is 0. If the result of a calculation is greater than 255, the value stored is 255. Division by 0 is not accepted.

The bottom line of the screen then displays the 'No. ?' query again, inviting you to amend another detail. If you have finished changing details, key '0' and press ENTER. The border of the screen then reverts to its normal grey colour. Note that, if you have changed row or column, the unit does not immediately move to its new position. It will appear there the next time the map is re-displayed. Similarly, if you have changed the symbol number, the new symbol is not seen until a change of map.

Installing BRIEFING

The listing of the subroutine appears in Chapter 9. The subroutine is a standard part of NASEBY and OMDURMAN, so all that is necessary is to type in the listing, as instructed in the sections dealing with setting up these programs.

Small alterations to the existing program are required when appending this subroutine to JUNGLE ATTACK. First of all load JUNGLE ATTACK. Next type in the subroutine listing. Now make the following amendments to the PHASE subroutine of the program, to allow it to call BRIEFING:

Line 2100: after 'AND mo<>114' insert 'AND mo<>98'.

Line 2200: after 'AND mo<>0' insert 'AND mo<>98'.

Line 2255 (new line) reads: '2255 IF mo=98 THEN BORDER 17:GOSUB 4800:REM BRIEFING'.

This completes the installation. Save the program in this form.

BRIEFING can be called in the Advance Phase and the Advance/Fire phase. You can call it as often as you wish for the same unit, and move the unit or fire it in the same phase as usual, either before or after briefing.

The Battle of Naseby decided the future of the monarchy in England. Since the outset of the Civil Wars, in 1642, the Royalist supporters of Charles I had conducted a ceaseless military campaign against the Parliamentary forces. Charles had gained a significant victory at Leicester in May 1645 and then decided to march to Oxford, which was under siege by the New Model Army of the Parliamentarians. In the meantime, the Parliamentarians, commanded by Sir Thomas Fairfax, had lifted the siege and were advancing north to engage Charles in battle. Charles, hearing of the approach of the New Model Army, decided to retreat to the Royalist garrison town of Newark. However, the Parliamentarians were nearer than he had imagined and, rather than retreat with them at his heels with consequent loss of morale among his troops, he decided to stand and engage them in battle. On the morning of 14 June, the Royalists were a few kilometres north of the village of Naseby. The countryside there is undulating, providing concealment for large numbers of men. The Royalists did not know exactly where the Parliamentarians were stationed, or their intended movements. The Royalists then deployed in line of battle on Dust Hill, straddling the road to Clipston and Market Harborough (level with Long Hold Spinney, Figure 9.1) along which they had intended to retreat. In fact, the Parliamentarians were to the south of the Royalists, closer to the village. After some discussion, Fairfax and his second-in-command Oliver Cromwell decided to deploy their army on the ridge immediately north of the village (level with Paisnell Spinney, Figure 9.1). Their regiment of dragoons was stationed behind Sulby Hedges, out of sight of the enemy.

To begin with, the Parliamentarians had deployed on the edge of the ridge, where they could be seen by the Royalists, but later withdrew a short distance. Most of their troops were thus hidden from the Royalists in a narrow depression that runs just behind the edge of the ridge. The Royalists believed that they were retreating. Charles did not suspect that the whole might of the New Model Army lay in wait behind the ridge.

Between the two armies lay the shallow valley of Broadmoor. This was marshy in places, and with patches of thorny scrub. The right flank of the Royalist cavalry led by Prince Rupert, moved forward, scouting for the

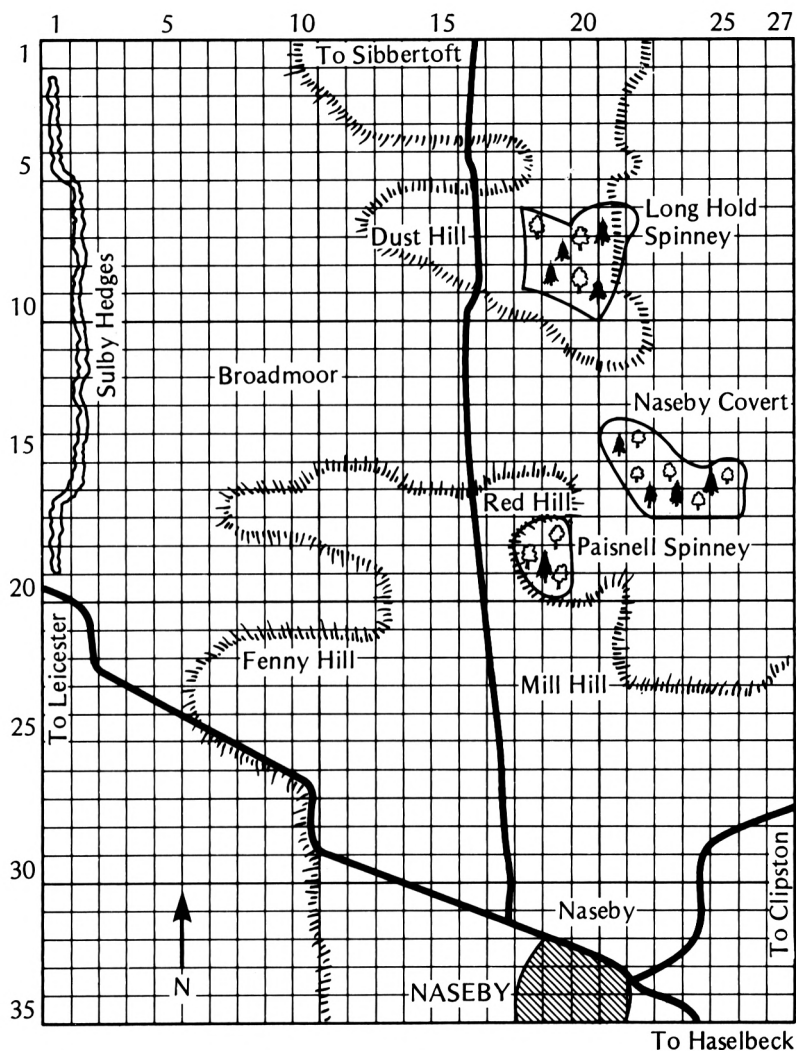


Fig 9.1 The Battlefield at Naseby (1 square = 100m).

Parliamentarians. As they crossed Broadmoor, Fairfax moved his troops forward over the ridge. The battle was about to begin.

Pike and shot

This battle belongs to the period of warfare popularly known as 'Pike and Shot' or, alternatively as 'Horse and Musket'. The armies on both sides consisted of regiments of cavalry and of infantry. Both sides also had artillery, but there is little evidence that these were used at Naseby or, if

they were, that they had any effect. The boggy nature of the ground and the relatively steep slopes would have made it difficult. For this reason, we have not included artillery in the game.



Fig 9.2 Royalist or Parliamentary Cavalry. The sash is coloured red for Royalists or buff for Parliamentarians.

The cavalrymen on both sides (Figure 9.2) were armed with matchlock pistols and swords. Having fired their pistols they halted to re-load them.

The infantry consisted of musketeers (Figure 9.3) and pikemen (Figure 9.4). The muskets, mainly the matchlock type which required the musketeer to carry a burning match to ignite the charge. The older, heavier models had to be supported on a rest when being aimed. Loading the musket was a lengthy procedure, taking about three and a half minutes! There were frequent misfirings, which makes five minutes a round a reasonable estimate of the firing rate. Some of the New Model Army were equipped with the newly developed and more efficient flint-lock muskets. Musketeers carried swords for use in close combat, when it was virtually impossible to reload and fire a musket.

The pikes were up to 5.5m long, tipped with a steel blade. In use, the pikes were held in the left hand and braced at the ground with the right foot. The array of pikes of a rank of pikemen presented an impenetrable barrier, especially for horses. Although the pike could be used as an offensive weapon, it was unwieldy. Some pikemen shortened their pikes to



Fig 9.3 A musketeer.



Fig 9.4 A pikeman.

make them more useful in close combat, but pikemen also carried swords for combat of this kind.

The combination of musketeers and pikemen was effective in that the musketeers could take cover below, between or behind the ranks of pikes. On occasions the unit would form a *schiltron*, which consisted of a square of pikemen, facing outward, with musketeers, officers and others in the centre. The composition of the infantry units of the two sides differed. The Parliamentarians had approximately two musketeers to one pikeman, which was the preferred ratio. Shortage of arms on the Royalist side resulted in their having about equal numbers of musketeers and pikemen. It was the custom to deploy a certain number of musketeers in front of the main line of battle. These were unprotected by pikemen and in open order. Their function was to act as skirmishers. By virtue of their exposed position, especially in the face of a cavalry charge, they were known as the *Forlorn Hope*.

The Parliamentary army also had dragoons (Figure 9.5). These were mounted infantrymen, armed with light muskets. The horses gave them mobility on the battlefield, but they normally dismounted to fire and reload their muskets. The Royalists were greatly outnumbered at Naseby. Against the 6000 infantry, 6500 cavalry and 1000 dragoons of the New Model Army, Charles had only about 4500 infantry and 4500 cavalry. Yet the morale of the Royalists was high, because of their recent victory at Leicester. In addition, their men were experienced veterans. The headstrong Prince Rupert inspired his cavalymen to deeds of daring. Unfortunately, his impetuous regiments left the field at a crucial moment and tried, unsuccessfully, to loot the Parliamentary baggage train (stationed just outside Naseby). By the time they returned to the field an

hour later, the Royalists had lost the battle. In this game, the Royalist player has the opportunity to keep Rupert's men under firmer control. Maybe this might alter the course of the battle in Charles' favour.



Fig 9.5 A dragoon.

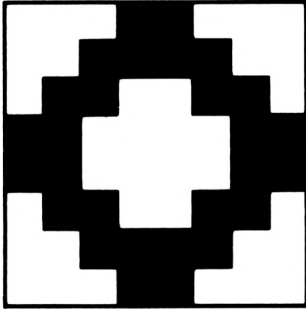
Hidden movement

This game incorporates line-of-sight hidden movement, as described in Chapter 7. The effect of this is that units behind hills, woods (including the Sulby Hedges) and the town of Naseby are out of sight of the enemy. A visit to the battlefield at Naseby is well worthwhile. Looking south from the road over Dust Hill, from the position of the Royalist battle lines, it is easy to imagine that the Parliamentary troops are there, hidden in the dip behind the opposite ridge. Where Fenny Hill slopes down toward the west, the Parliamentary left flank would just be visible near Sulby Hedges. Comparing the screen displays with observations we have made on the site of the battle confirms that the subroutine used in this program is a good simulation.

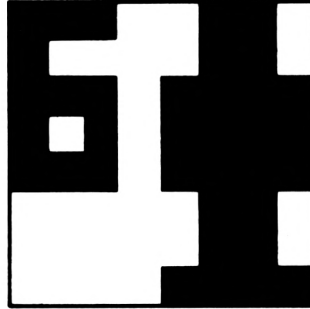
Naseby is only 15km from junction 18 of the M1, on the Daventry–Market Harborough road (B4036). The site of the battle is beside the Naseby–Sibbertoft road (Grid ref: SP 684 802). A monument commemorates a cavalry charge led by Cromwell, which largely decided the outcome of the battle.

The game

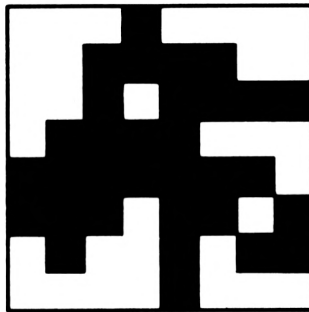
When the game begins, the units of both sides are already deployed in their historical positions as at the start of the battle. From then on, players are free to move and engage the units in battle in any way they please, subject to the rules below and to any other rules the players themselves may decide



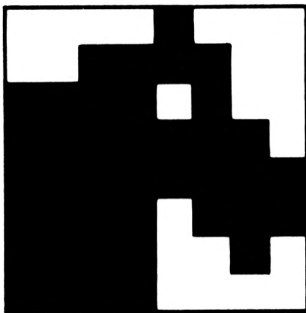
202



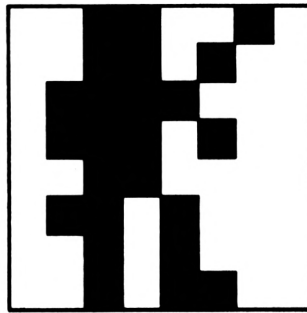
240



244



254



255

Fig 9.6 Symbols used to represent units of the Parliamentary Army. 202, Fairfax; 240, Infantry regiment (6-Col. Hammond's); 255, Forlorn Hope; 244, Cavalry regiment (2-Col. Vermuyden); 254, Dragoons. Skippon, Ireton, Cromwell and the baggage train are represented by S, I, C, and #. Other cavalry under Ireton and Cromwell are like 244 but marked with an I or C.

upon. We have suggested a minimal set of rules sufficient for an interesting game, but the game program is flexible enough to allow players to adopt their own set of rules without the need to alter the program.

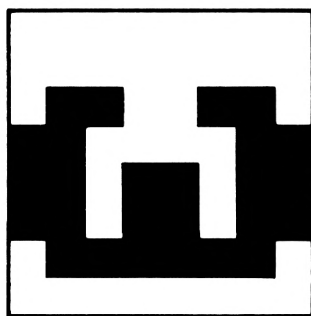
Time scale The game consists of 24 game-turns, each representing five minutes of real time.

Map scale The map (Figure 9.1) is divided into squares, the side of a square being equivalent to 100m.

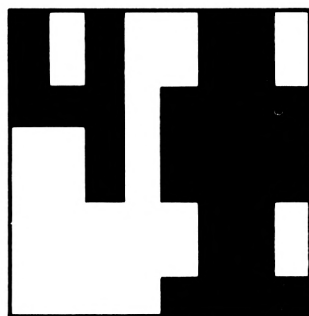
Unit scale The number of men represented by a unit varies. The leaders, Charles I and Fairfax, are each represented by a single unit. They do not take part in combat. The Parliamentary army consists of eight infantry units, each of 750 men. There are also two units of Forlorn Hope, of 150 men each. The cavalry comprise 21 units, each representing squadrons of 300 men and horses. The dragoons are represented by three units, each of 330 men. The baggage train is a single unit, guarded by 100 musketeers.

On the Royalist side there are eight regiments of infantry, each of 600 men, two detachments of Forlorn hope, of 100 men each, and 13 squadrons of cavalry, of 300 men each. The baggage train is undefended.

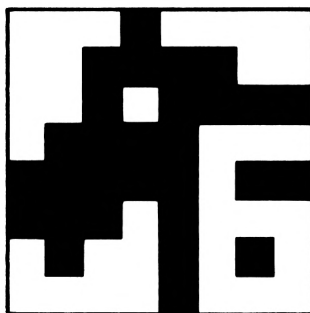
Further details of the armies and the names of the individual units are given in Tables 7 and 8. The symbols for the units are shown in Figures 9.6



189



238



248

Fig 9.7 Symbols used to represent units of the Royalist Army. 189, King Charles; 238, Infantry regiment (4-Sir Henry Bard); 248, Cavalry Regiment (6-Northern Horse); Astley and Prince Rupert are represented by A and R. Forlorn Hope and baggage train as in Fig 9.6

Table 7 Parliamentary Army

Army name: Fairfax's

Paper no. 0: Pen no. 1: 7 types of unit

Unit no.	Type name and Commander	Type no.	Symb	Status	NULL	Disorg	Str	Mor	Row	Col
<i>Leader</i>										
1	Sir Thos. Fairfax	1	202	4	0	0	1	9	20	11
<i>Infantry regiments:</i>										
<i>(2 musketeers: 1 pike):</i>										
2	Sir Philip Skippon (S)	2	83	4	0	0	250	9	18	8
3	Sir Hardress Waller (1)	2	235	4	0	0	250	7	18	9
4	Col. Pickering (2)	2	236	4	0	0	250	7	18	10
5	Col. Montague (3)	2	237	4	0	0	250	7	18	11
6	The Lord Gen. (Fairfax) (4)	2	238	4	0	0	250	7	18	12
7	Lt.-Col. Pride (5)	2	239	4	0	0	250	7	19	9
8	Col. Hammond (6)	2	240	4	0	0	250	7	19	10
9	Col. Rainsborough (7)	2	241	4	0	0	250	7	19	12
<i>Infantry (all musketeers):</i>										
10	Forlorn Hope	4	255	4	0	0	50	8	17	7
11	Forlorn Hope	4	255	4	0	0	50	8	17	13
<i>Cavalry regiments:</i>										
12	Col. Butler (1)	5	243	4	0	0	100	7	18	2
13	Col. Butler (1)	5	243	4	0	0	100	7	18	3
14	Col. Vermuyden (2)	5	244	4	0	0	100	7	18	4
15	Col. Vermuyden (2)	5	244	4	0	0	100	7	18	5
16	Commissary-Gen. Ireton (1)	5	73	4	0	0	100	9	18	6
17	Commissary-Gen. Ireton (1)	5	252	4	0	0	100	8	18	7
18	Col. Riche (3)	5	245	4	0	0	100	7	19	3
19	Col. Riche (3)	5	245	4	0	0	100	7	19	4
20	Col. Fleetwood (4)	5	246	4	0	0	100	7	19	5
21	Col. Fleetwood (4)	5	246	4	0	0	100	7	19	6
22	Association Horse (5)	5	247	4	0	0	100	7	19	7
23	Col. Whalley (6)	5	248	4	0	0	100	7	18	13
24	Col. Whalley (6)	5	248	4	0	0	100	7	18	14
25	Sir Robert Pye (7)	5	249	4	0	0	100	7	18	15
26	Sir Robert Pye (7)	5	249	4	0	0	100	7	19	15
27	Lord General (Oliver Cromwell) (C)	5	67	4	0	0	100	9	18	16
28	Lord General (C)	5	253	4	0	0	100	8	18	17
29	Col. Sheffield (8)	5	250	4	0	0	100	7	19	13
30	Col. Sheffield (8)	5	250	4	0	0	100	7	19	14
31	Col. Fienne (9)	5	251	4	0	0	100	7	19	16
32	Col. Rossiter (blank)	5	242	4	0	0	100	7	20	17
<i>Dragoons:</i>										
33	Col. Okey	6	254	5	0	0	110	8	12	1
34	Col. Okey	6	254	5	0	0	110	8	14	1
35	Col. Okey	6	254	5	0	0	110	8	16	1
<i>Baggage train:</i>										
36	(Guarded by musketeers)	7	35	4	0	0	33	7	28	13

Symb=symbol number; NULL=null detail; Str=strength/3; Disorg=disorganisation level; Mor=moral level.

The digits or letters in brackets following the names of each unit commander are those used to identify the symbols on the screen.

and 9.7. On the battlefield, men of either side wore similar clothing and armour. Colours of coats and other garments varied according to regiment,

so *both* sides included men wearing red coats, blue coats or coats of other colours. To avoid confusion, soldiers wore a sash; red for the Royalist army and buff for the Parliamentarians. In the game, the unit symbols are on backgrounds of these two colours.

Table 8 Royalist Army

Army name: Charles'
 Paper no. 3: Pen no. 4: 7 types of unit

Unit no.	Type name and Commander	Type	Symb	Status	NULL	Disorg	Str	Mor	Row	Col
<i>Leader:</i>										
1	King Charles I	1	189	6	0	0	1	8	4	10
<i>Infantry Brigades:</i>										
<i>(1 musketeer: 1 pike):</i>										
2	Sir Bernard Astley (A)	3	65	6	0	0	200	9	8	9
3	Sir Bernard Astley (1)	3	235	6	0	0	200	8	8	10
4	Sir Bernard Astley (2)	3	236	6	0	0	200	8	6	9
5	Sir Henry Bard (3)	3	237	6	0	0	200	8	8	11
6	Sir Henry Bard (4)	3	238	6	0	0	200	8	6	10
7	Col. Sir George Lisle (5)	3	239	6	0	0	200	8	8	12
8	Col. Sir George Lisle (6)	3	240	6	0	0	200	8	6	11
9	King's Lifeguard of Foot/ Pr. Rupert's Regt. of Foot (blank)	9	234	6	0	0	50	9	5	10
<i>Infantry (all musketeers):</i>										
10	Forlorn Hope	4	255	6	0	0	33	8	9	5
11	Forlorn Hope	4	255	6	0	0	33	8	9	16
<i>Cavalry regiments:</i>										
12	Pr. Rupert's Lifeguard (R)	5	82	6	0	0	100	9	8	6
13	Pr. Rupert's Regiment of Horse (1)	5	243	6	0	0	100	9	8	7
14	Queen's/Pr. Maurice's (2)	5	244	6	0	0	100	8	8	8
15	Earl of Northampton (3)	5	245	6	0	0	100	8	7	7
16	Sir William Vaughan (4)	5	246	6	0	0	100	8	7	8
17	Sir Thomas Howard (5)	5	247	6	0	0	100	8	7	10
18	Sir Marmaduke Langdale (L)	5	76	6	0	0	100	9	8	13
19	Northern Horse (6)	5	248	6	0	0	100	6	8	14
20	Northern Horse (6)	5	248	6	0	0	100	6	8	15
21	Northern Horse (6)	5	248	6	0	0	100	6	7	13
22	Northern Horse (6)	5	248	6	0	0	100	6	7	14
23	Northern Horse (6)	5	248	6	0	0	100	6	7	15
24	Newark Horse (blank)	5	242	6	0	0	100	8	5	9
25	King's Lifeguard of Horse (blank)	5	242	6	0	0	100	9	5	11
<i>Baggage train:</i>										
26	(Unguarded)	7	35	6	0	0	0	0	3	6

Symb=symbol number; NULL=null detail; Str=strength/3; Disorg=disorganisation level; Mor=moral level.
 The digits or letters in brackets following the names of each unit commander are those used to identify the symbols on the screen.

Sequence of play

In each game turn there are two player-turns, the Parliamentarian side playing first, the Royalist side playing second. Each player-turn consists of four phases.

1 *Advance Phase* Units are advanced or change direction.

2 *Charge Phase* Units that moved during the Advance phase are allowed to move additional squares, provided that they finish this phase on a square already occupied by an enemy unit.

3 *Battle Phase* Units that are one or two squares from an enemy unit may exchange fire. Since the range of a pistol is only one square, cavalry cannot fire unless they are on a square adjacent to an enemy unit. Infantry have muskets, so can fire at a range of one or two squares. If a unit is on a square occupied by an enemy unit, the two units are in close combat or *mêlée*. In the first turn of a *mêlée*, the units fire pistols or muskets, if so armed. Subsequently, they fight with swords.

4 *Rout/Rally Phase* After a round of combat, any unit that has suffered badly may be required to rout (retreat in disorder). Later it may have recovered sufficiently to rally and return to the battle. This phase, gives *both* sides the opportunity to rout or rally such units. So that players do not see the disposition of opposing troops they should change places at the end of each Rout/Rally phase. After the Rout/Rally phase there is a delay while the computer 'hides' all units that are out of sight of all enemy units. This is in readiness for displaying the map next turn. In the first few turns of the game, hiding may take several minutes, during which the message 'Making ready' is displayed. The routine takes much less time as the game develops, because the distances between units become shorter and fewer units are out of sight.

Finally, a complete roster of the army is displayed, showing the current status and location of each unit. Players should change places after this roster.

Method of play

Unit details When a unit has been selected to advance or engage in combat, or when a roster is called for (see Chapter 6), the current status of the unit is displayed at the bottom of the screen. The details displayed are:

Unit symbol.

Arrow (green) to show the direction the unit is facing.

Morale level (blue, see later).

Routing status, shown as a yellow 'R' if being routed.

Disorganisation level (light green, see later).

Strength (numbers of men, in dark cyan). Note that, owing to the way the strength is stored, a single leader is displayed as '3', while 100 men are displayed as '99'.

Advance Phase and Charge Phase Units are advanced and map sections are changed as described in Chapter 6. In this game, the rostering routine

displays unit details of both armies. This is because the size of a unit, and the direction in which it is facing, cannot be shown on the map, yet would be clearly visible to observers on both sides. However additional rules apply, the most important of which is that governing the *direction* of advance. When the 'A' key is pressed, the unit's details appear as usual. These include an arrow which points north, east, south or west. The unit may only advance in the direction shown. At the start of the game, all Royalist units are facing south. All Parliamentary units are facing north, except dragoons, which face east. With certain exceptions, which will be explained later, units take one turn to change direction, during which they cannot advance or charge. Change of direction is made during the Advance Phase only, by using the BRIEFING procedure, as described later.

It is advisable to have pen and paper to hand for recording the row and column of those squares in which a *mêlée* is in progress. It is also worthwhile recording whether a unit entered an enemy-occupied square as a result of advancing or of charging, for this makes a difference to the casualties on both sides.

The method of mounting or dismounting a dragoon is described later.

Battle Phase The method of play is the same as described in Chapter 3 and combat is resolved automatically. If opposing units are on the same square, this is close combat or *mêlée*. Move the cursor to the *mêlée* square, and press the space bar. The border of the screen turns red and details of the attacking unit are displayed below the screen. If this is the first turn of the *mêlée*, press 'M'. The border turns yellow and combat is resolved as if loaded pistols and muskets were used. In subsequent turns, press the space bar a second time (instead of pressing 'M'), to resolve combat using swords. The result of combat is displayed below the screen as a flashing message. First we see the losses of the attacking unit (only the first letter of the army name, F or C, is given). Then we see the losses of the defending unit, expressed as two numbers. The first number is the loss, the second is the size of the unit before the loss occurred. Note that *both* sides take part simultaneously in combat, so there are likely to be losses on both sides. If one unit has just charged into the square, or has entered it from the flank or rear, note the numbers of losses on paper (see later). If a unit loses 10% or more of its strength in one round of combat, the unit has to be routed, as explained later. If the unit has to be routed, a message to this effect is displayed after the losses have been reported. Possibly both units may have to be routed.

Rout/Rally Phase There are two sub-phases, one for each army, since either or both armies may be routed as a result of combat. Units which are routed will be distinguished by the yellow 'R' displayed in their unit details. Such units are moved in the same way as in the Advance phase, and for the same distance, but special rules apply.

Rules of play

Some of these rules are built into the program, so take effect automatically and are not easily altered. The majority of the rules are optional and can be varied at the discretion of the players.

Movement

Moving Units This takes place during the Advance, Charge and Rout/Rally phases. Special rules apply to routing, as explained in the next section. The maximum movement allowed is controlled by the program.

Each unit is allocated a number of movement points each phase:

<i>Unit type</i>	<i>Allowance during</i>	
	<i>Advance or Rout/ Rally Phases</i>	<i>Charge Phase</i>
1-4 Infantry and dismounted dragoons	4	4
5-6 Cavalry and mounted dragoons	8	2
7 Baggage train	3	0

The movement costs for each square of terrain are:

<i>Terrain</i>	<i>Points</i>
Open, track	2
Hedge, rough ground, slopes	3
Woods, Naseby village	9

The slopes are the squares at the edge of the high ground. The rough ground consists of isolated squares, not specially marked on the map. These represent specially boggy or thorny areas which hamper the movement of troops unexpectedly. The value of 9 for woods and Naseby means that units cannot enter these areas.

The following optional rules apply for movement:

Units other than a leader may not move onto or through a square occupied by a friendly unit consisting of more than 100 men.

Units may not move through a square occupied by an enemy unit consisting of more than 100 men, but may advance or charge on to such a square.

A square in which there are enemy pikemen may only be entered from the flanks (sides) or rear.

Units that have not moved in the Advance phase, may not move in the Charge phase. A unit may move in the Charge phase only if it finishes on a square occupied by an enemy unit. The effect of these two rules is that, at the beginning of the Advance phase, a unit must be at least two squares from the square it intends to charge, thus giving it the distance in which to gain speed.

Units must move in the direction shown by their arrow, with the following exceptions: leaders, Forlorn Hope, units leaving a mêlée, routed units.

If a unit is to change direction, the BRIEFING routine is used during the Advance phase. However, to allow a unit to make a diagonally forward movement without (unrealistically) having to change direction twice, the movements shown in Figure 9.8 are allowed without change of direction, provided that they are completed in one turn.

Moving routed units Routed units must move as far as they can in a direction away from the nearest enemy units (ignore the direction arrows).

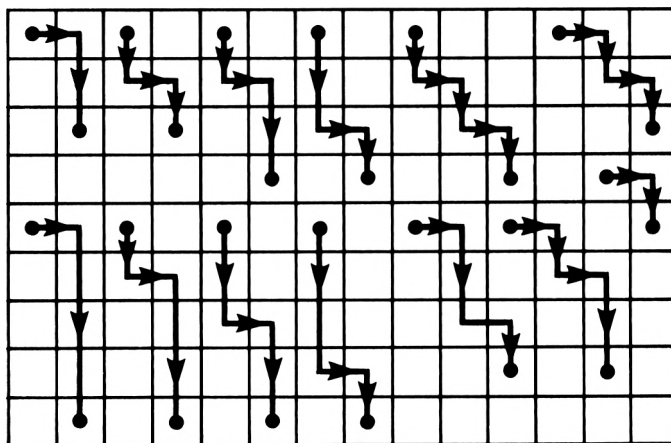


Fig 9.8 Diagonal movements allowed without change of direction. The drawings show paths for a unit initially facing south and moving diagonally south and east. Similar paths are allowed for units facing in other directions.

They cannot pass through squares occupied by enemy units, or through squares occupied by friendly units of strength greater than 100. Note that passing through or close by friendly units may cause the friendly units to lose morale (see later). If a routed unit is unable to move its full distance owing to squares being occupied by other units, or being wood or village squares, or squares at the edge of the map, they gain three disorganisation points. If you are not taking disorganisation into account in your game they are considered to have surrendered, and are removed from the game.

Battle

Combat resolution is controlled by the program. *The following optional rules apply.*

Units may not fire into a square on which a mêlée is in progress (it is very useful to keep a list of such squares). Units engaged in a mêlée may not fire out of the square.

Units at a range of two squares from their target may not fire *through* a square occupied by other units, either friends or enemies.

When entering a square to engage in a mêlée, or engaging an enemy on its flanks or rear, make a note of this fact and the row and column of the square concerned, for use after automatic combat resolution (see BRIEFING).

In each Battle phase, combat *must* be resolved on all squares on which there is a mêlée. In addition, firing between adjacent squares, or at a range of two squares may occur at the option of the player.

If the strength of a unit is reduced to less than 70% of its original strength when its morale level is 7 or less, or to less than 60% of its original strength when its morale is 8 or less, that unit is considered to have deserted or surrendered, and must be removed from the game (see later). For the first turn at which a unit charges into a square to engage in mêlée, the casualties

of that unit are reduced by 20%. The casualties of the enemy unit are increased by 20%. The way to do this is explained later.

The previous rule also applies for the first turn of a *mêlée* in which the square was entered from the flanks or rear. If the square was charged at the flanks or rear, the effects of charging and the direction of entry are combined, giving a 40% decrease and increase of casualties.

Morale

All rules in this section are optional. The morale of a unit determines its resilience under battle conditions. In particular it governs whether or not the unit is likely to desert or surrender when under pressure. The units in this game each begin with a morale level dependent mainly on events which occurred before the period covered by the game. As already described, this effects the maximum number of casualties that a unit will suffer before it deserts. Although the unit details display the current morale level, this is not taken into account in the program. Thus players are free to adopt their own morale rules. To simplify playing the first game or two, readers may prefer to let morale remain at its initial level for the duration of the game. But in a real battle, morale may fall or be boosted as a result of events occurring during combat. The maximum morale level is 9. Events which would cause it to become greater than 9 are ignored. Below are some suggestions for factors that may affect morale. Readers may add their own ideas to this list:

<i>Event</i>	<i>Change in morale</i>
Enemy leader captured or surrenders	+2
If routed, own leader or friendly unrouted unit present on an adjacent square	+1
Routing enemy unit within 1 square	+1
If routed and on higher ground than the nearest enemy unit	+1
Charging into <i>mêlée</i> from higher ground	+1
Being charged from higher ground	-1
If routed and on lower ground than the nearest enemy unit	-1
Routed friendly unit on same or adjacent square	-1
Loss of 10% of strength in 1 turn	-1
Own baggage train captured	-1
Own leader captured or surrenders	-2

The surrender or capturing of the Commander-in-Chief, Charles or Fairfax, brings the game to an end (see *Victory*, later). Other leaders of the Royalists to which the morale changes above apply are Prince Rupert (cavalry of right flank), Astley (infantry), and Langdale (cavalry of left flank). When these leaders are captured or surrender, all units under their command are affected, losing two points each. On the Parliamentary side, the leaders are Ireton (cavalry of left flank), Skipton (infantry), Cromwell (cavalry of right flank) and Okey (Dragoons). When one of these leaders is captured or surrenders, all enemy units within six squares gain

two morale points. When a baggage train is captured, all friendly units suffer a loss in morale. All enemy units within six squares gain morale.

Morale should be checked whenever the events described above occur and alterations made to morale levels, using the BRIEFING routine (see later).

If morale falls below 5, the unit is routed during the Rout/Rally phase. It is then unable to move or fire, but can be attacked. Any combat losses inflicted on these attackers are restored by BRIEFING. Morale is checked during the Rout/Rally phase in each subsequent turn according to the table above and then increased by an additional one. If it rises above five, the unit is rallied. Its routed status is reset to normal, using BRIEFING, and it can move again and return to the battle next turn. If morale falls below zero, the unit is eliminated from the battle.

Disorganisation

This reflects the extent to which the troops maintain their correct formation and thus maintain their highest combat efficiency. As with morale, disorganisation level is displayed as one of the unit details, but does not affect any of the routines of the program. Players are free to disregard disorganisation, to adopt the rules below, or to apply their own rules.

Suggested disorganisation rules are as follows. They are applied after combat, during the Rout/Rally phase. Units begin the game with zero disorganization points, but gain points for the following reasons:

Losing 5% or more casualties in a turn	1
Each turn in mêlée	1
Attacked from flank or rear	1
Routed	2
Charging but failing to enter the enemy square (due to rough ground or slope)	2
Unable to rout when required	3
Attacked immediately after having changed direction	3
Attacked immediately after having changed direction	3

A unit with more than zero disorganisation points may lose two points in any turn in which they do not advance or fire. The effect of disorganisation points on firing and mêlée is to reduce the number of casualties inflicted on the enemy by 20% for each point. Thus a unit with five points inflicts no casualties.

Reforming units

The following rules are optional. Two units of the same type can be reformed into a single unit. The total strength of the reformed unit must not exceed 750. The units must be on adjacent squares and neither must be engaged in mêlée. Two routed units may not reform as one unit, but a routed unit may be reformed with a non-routed unit. Reforming takes place during the Advance phase only, and the reformed unit may not move or fire until the following game-turn. The BRIEFING routine (see below) is used for reforming. The strength of one unit is altered to the total strength of the two units. The strength of the other unit is altered to zero, and it is

Surrender or desert The following sequence removes a unit from the game. Key '2' in response to 'No.?''. The value displayed is the current row. Key '0' in response to the next 'No.?''. The value displayed is the current column. Key '0' in response to 'No.?''. Finally, key '0' in response to 'No.?''.

Changing direction Key '5' in response to 'No.?''. If the value displayed is 0 to 3 the unit is visible and the direction is given 0=north, 1=east, 2=south and 3=west. If the unit is hidden, the value displayed is 4 more than those above. If the unit is routing, the value displayed is 8 more. To make one quarter-turn clockwise, the usual response is to key '+1'. However, if the unit is facing west (the value displayed is 3, 7, 11 or 15) key '-3'. Finally, key '0' in response to 'No.?''.

Routing Key '5' in response to 'No.?''. The number displayed shows direction, hidden and rout status (as explained above). If less than 8 the unit is not routing. Key '+8' to make the unit rout. Key '-8' to cancel routing. Finally key '0' in response to 'No.?''.

Disorder Key '7' in response to 'No.?''. The value displayed is the disorder level. Key a number preceded by '=', '+' or '-', as convenient, to set disorder to its new value. Finally, key '0' in response to 'No.?''.

Strength Key '8' in response to 'No.?''. The value displayed is *one third* of the actual strength. Key a number, preceded by '+' or '-', equivalent to one third of the gain or loss of strength. Finally, key '0' in response to 'No.?''.

Morale Key '9' in response to 'No.?''. Then proceed as for disorganisation.

Victory

It is suggested that the game ends when the supreme leader of one side, (Charles or Fairfax), is captured or surrenders. If the enemy leader is on a square by himself or on the same square as a routed enemy unit, capture is effected by moving an unrouted unit on to that square. If the enemy leader is on a square occupied by an unrouted enemy unit, the unit may be engaged in combat. The fate of the leader is then in the hands of the enemy unit. If the unit is routed or surrenders (because of excessive loss of strength or lowness of morale), the leader is considered to have been captured or to have surrendered.

Saving the game

You are given the option of saving the game to tape or disk at the end of every player-turn. To do this, follow the instructions in Chapter 6.

Setting up

It is assumed that you have already typed in the program of JUNGLE ATTACK (Chapter 6), and have played it several times. If so, you have a well-tested program that needs only a little alteration to turn it into the NASEBY program. If you have *not* already typed in JUNGLE ATTACK, type it in now, according to the instructions and listing given in Chapter 6. Note that you do not need the cipher program, JUNGLE C. When typing in JUNGLE ATTACK, you can save time by omitting the main program (lines

10–200) and certain subroutines that are not required for NASEBY. These are:

1300–1460 DEPLOY
 1470–1570 UNITLINE/ATTACK
 2320–2380 PROX
 2390–2850 CLOSE
 3420–3920 FIRE
 3990–4270 MORTAR
 4280–4370 PROJECTILE
 4610–4630 DIGIN
 4640–4760 SYMBOLS
 4770–4790 HIDECOVER (if present)

When omitting these routines, simply omit to type them in, but *do not renumber* the program.

If you have already a working version of JUNGLE ATTACK, delete the main program (lines 10–200) and the subroutines listed above. *Do not renumber* the program.

Having obtained the bulk of the program in one of the ways described above, you need to make the following additions and amendments:

1 Type in the main program, NASEBY:

```

10 REM ** NASEBY **
20 PAPER 0:PEN 1:INK 1,24:CLS
30 MEMORY 28823:CLS:GOSUB 3030:INK 0,15:gturn=2:cpf=3
:cevf=2000:DIM ho(maxu):REM START
40 turn=PEEK(30000)+1:POKE 30000,turn
50 numarmy=turn-INT((turn-1)/PEEK(30001))*PEEK(30001)

60 na=numarmy:cp=0:fp=0:m$="Advance":GOSUB 2060:REM PHASE
70 na=numarmy:cp=1:fp=0:m$="Charge":GOSUB 2060:REM PHASE
80 na=numarmy:cp=0:fp=1:m$="Battle":GOSUB 2060:REM PHASE
90 na=numarmy:cp=0:fp=0:m$="Rout":GOSUB 2060:REM PHASE
100 na=numarmy:IF na=1 THEN na=2:ELSE na=1
110 cp=0:fp=0:m$="Rout":GOSUB 2060:REM PHASE
120 p1=numarmy:IF p1=1 THEN p2=2:ELSE p2=1
130 GOSUB 5230:GOSUB 5340:REM HIDE0BST CONCEAL
140 na=numarmy:GOSUB 2890:REM TURNEND
150 IF turn=49 THEN 180
160 IF mo=240 THEN 40
170 GOSUB 3000:REM SAVE
180 PAPER 0:PEN 1:MODE 1:LOCATE 2,2:PRINT"Game finished"
190 GOTO 190

```

2 Amend the ADVANCE routine by deleting lines 1080 and 1090. Amend lines 1110–1120 as follows:

```

1110 bu=aa(na)+2+9*u:all=4:IF PEEK(bu)>4 THEN all=8:IF PEEK(bu)=7 THEN all=3
1120 IF cp THEN all=4:IF PEEK(bu)>4 THEN all=2:IF PEEK(bu)=7 THEN f$="No charge":GOSUB 3960:RETURN

```

- 3 Delete lines 1220 and 1240 from the DISPARMY subroutine.
- 4 Type in this version of the UNITLINE subroutine:

```

1470 bu=aa(ny)+9*u+2:REM UNITLINE/NASEBY ***
1480 PEN #scr,pe(ny):PAPER #scr,pa(ny):LOCATE #scr,vx
,vy:PRINT #scr,CHR$(PEEK(bu+3));
1490 PEN #scr,12:PAPER #scr,11:PRINT #scr,CHR$(1);CHR
$(11-2*(PEEK(bu+4) AND 1)-(PEEK(bu+4) AND 2)/2);
1500 SOUND 1,100,6
1510 PEN #scr,10:PRINT #scr,MID$(STR$(PEEK(bu+8)),2);
1520 IF(PEEK(bu+4) AND 8) THEN PEN #scr,1:PRINT #scr,
"R ";ELSE PRINT #scr," ";
1530 PEN #scr,13:PRINT #scr,MID$(STR$(PEEK(bu+6)),2);
" ";
1540 PEN #scr,8:PRINT #scr,MID$(STR$(PEEK(bu+7)*cpf),
2);
1550 RETURN

```

- 5 Modify the PHASE subroutine:

```

2060 CLS #1:CLS #2:CLS #3:CLS #4:GOSUB 1950:REM INITI
AL:REM PHASE ***
2070 FOR j=1 TO PEEK(30001):FOR k=1 TO maxu:d(j,k)=0:
NEXT:NEXT
2080 PRINT#1,"TURN"INT((turn-1)/gturn+1)m$;
2090 f$=aname$(na)+" army":GOSUB 3960:REM MESSAGE
2100 CLS #3:mo=0:WHILE mo<>32 AND mo<>97 AND mo<>98 A
ND mo<>114: GOSUB 1580:WEND:REM COMMAND
2110 IF wind=0 THEN GOTO 2140
2120 IF xc4=3 AND yc4=3 THEN CLS #0:CLS# 1:CLS #2:CLS
#3:CLS #4:RETURN
2130 GOSUB 2000:GOTO 2100:REM NEWMAP
2140 u=1:usq=0:nz=1
2160 uf=usq:uz=u:xz=xm:yz=ym:GOSUB 2270:u=uz:xf=xm:yf
=ym:usq=uf:REM FINDUNIT
2170 IF uf=0 THEN IF nz=2 THEN PRINT#3,"Roster comple
te";t=TIME:WHILE TIME<t+250:WEND:CLS #3:GOTO 2100:EL
SE nz=2:u=1:usq=0:GOTO 2160
2180 xp=xc:yp=yc-1
2190 ny=nz:scr=3:vx=1:vy=1:GOSUB 1470:t=TIME:WHILE TI
ME<t+400:WEND:REM UNITLINE/NASEBY
2200 IF mo=114 AND nz=na THEN mo=1000:WHILE NOT(mo=32
AND fp=1) AND mo<>97 AND mo<>98 AND mo<>114 AND mo<>
0:GOSUB 4380:WEND
2210 IF mo=0 THEN mo=114
2220 IF mo=114 OR nz<>na THEN 2160
2230 IF mo=32 AND fp=1 THEN BORDER 3:GOSUB 4950:REM B
ATTLE
2240 IF mo=97 THEN BORDER 1:POKE bu+4,PEEK(bu+4) AND
127:GOSUB 1000:REM ADVANCE
2250 IF mo=98 THEN BORDER 17:GOSUB 4800:REM BRIEFING
2260 BORDER 13:GOTO 2100

```

- 6 Amend line 3080 in the START subroutine:

```

3080 LOAD "NASEBYD"

```

7 Modify line 3300 of the PATCH subroutine:

```
3300 IF ((PEEK(bu+4) AND 4) AND nz<>numarmy) THEN 328
0
```

8 Type in this version of the SYMBOLS routine, noting that the lines are numbered in fives:

```
4640 CALL 29200,28824,234:REM SYMBOLS/NASEBY
4650 SYMBOL 234,24,24,60,60,60,24,24,60
4655 SYMBOL 235,70,70,79,79,79,6,6,15
4660 SYMBOL 236,230,38,79,143,239,6,6,15
4665 SYMBOL 237,230,38,239,47,239,6,6,15
4670 SYMBOL 238,166,166,239,47,47,6,6,15
4675 SYMBOL 239,230,134,239,47,239,6,6,15
4680 SYMBOL 240,230,134,239,175,239,6,6,15
4685 SYMBOL 241,230,166,47,47,47,6,6,15
4690 SYMBOL 242,16,60,47,127,255,239,79,15
4695 SYMBOL 243,16,60,47,125,253,237,77,13
4700 SYMBOL 244,16,60,47,120,254,237,75,8
4705 SYMBOL 245,16,40,47,120,254,232,78,8
4710 SYMBOL 246,16,60,47,122,250,232,78,14
4715 SYMBOL 247,16,60,47,120,251,232,78,8
4720 SYMBOL 248,16,60,47,120,251,232,74,8
4725 SYMBOL 249,16,60,47,120,250,238,78,14
4730 SYMBOL 250,16,60,47,120,250,232,74,8
4735 SYMBOL 251,16,60,47,120,250,232,78,8
4740 SYMBOL 252,16,60,47,120,253,237,77,8
4745 SYMBOL 253,16,60,47,120,250,235,74,8
4750 SYMBOL 254,8,60,244,254,255,247,242,240
4755 SYMBOL 255,50,52,120,116,48,104,40,44
4760 RETURN
```

9 Add the following new subroutines:

```
4800 PAPER #3,5:PEN #3,4:REM BRIEFING ***
4810 CLS #3:detail=1:WHILE detail>0
4820 CLS #3:detail=-1:WHILE detail<0 OR detail>9:INPUT
T #3,"No. ";detail$:detail=VAL(detail$):WEND
4830 IF detail=0 THEN 4940
4840 char=0:value=-1:WHILE (char<>42 AND char<>43 AND
char<>45 AND char<>47 AND char<>61) OR (value<0 OR v
alue>255):PRINT #3,"="PEEK(bu+detail-1);CHR$(1);CHR$(
8);
4850 value$="":WHILE value$="":INPUT #3,value$:WEND
4860 char=ASC(LEFT$(value$,1)):value=VAL(MID$(value$,
2))
4870 WEND
4880 IF char=42 THEN value=PEEK(bu+detail-1)*value
4890 IF char=43 THEN value=PEEK(bu+detail-1)+value
4900 IF char=45 THEN value=PEEK(bu+detail-1)-value
4910 IF char=47 AND value>0 THEN value=PEEK(bu+detail
-1)/value
4920 value=ROUND(value):value=MIN(255,value):value=MAX(0,value)
4930 POKE bu+detail-1,value
4940 WEND:PAPER #3,11:CLS #3:RETURN
4950 GOSUB 4380:REM BATTLE ***
```

```

4960 IF mo<>0 THEN GOSUB 1180:REM PIP
4970 IF mo=32 OR mo=109 THEN PRINT CHR$(7);:GOTO 5000

4980 GOSUB 1660:REM CURSOR0
4990 xc=xn:yc=yn:GOTO 4950
5000 IF xf=xm AND yf=ym THEN cevt=3:IF mo=109 THEN ce
vt=2
5010 IF d(na,u) THEN f$="Already played":GOSUB 3960:R
ETURN
5020 pb=PEEK(aa(na)+6+u*9) AND 8:IF pb THEN f$="Unit
routing":GOSUB 3960:RETURN
5030 BORDER 24:dx=xm-xf:dy=ym-yf:ax=ABS(dx):ay=ABS(dy
):IF ax<2 AND ay<2 THEN 5050
5040 IF mo=32 THEN GOSUB 4420:IF fb>1 THEN f$="Out of
sight":GOSUB 3960:RETURN:REM LOS/MESSAGE
5050 IF na=1 THEN p1=1:p2=2:ELSE p1=2:p2=1
5060 d(na,u)=1:uz=1:uf=1:nz=p2:xz=xm:yz=ym:GOSUB 2270
:REM FINDUNIT
5070 IF uf=0 THEN f$="No enemy":GOSUB 3960:RETURN
5080 ra=INT(SQR(ax*ax+ay*ay))
5090 IF ra>2 THEN f$="Out of range":GOSUB 3960:RETURN
5100 IF ra=2 THEN cevt=0
5110 IF ra=1 THEN cevt=1
5120 cu1=u:cu2=uz:bu=aa(p1)+2+cu1*9:be=aa(p2)+2+cu2*9
:cp1=PEEK(bu+7):cp2=PEEK(be+7)
5130 GOSUB 2860:REM RESCOM
5140 dcp1=MIN(ROUND((dcp1*(1+FNranorm/50))/cevf),cp1)
:POKE bu+7,cp1-dcp1
5150 dcp2=MIN(ROUND((dcp2*(1+FNranorm/50))/cevf),cp2)
:POKE be+7,cp2-dcp2
5160 f$=LEFT$(aname$(p1),1)+" loss"+STR$(dcp1*cpf):GO
SUB 3960:REM MESSAGE
5170 t=TIME:WHILE TIME<t+1000:WEND
5180 f$=LEFT$(aname$(p2),1)+" loss"+STR$(dcp2*cpf)+"/
"+MID$(STR$(cp2*cpf),2):GOSUB 3960:REM MESSAGE
5190 t=TIME:WHILE TIME<t+1000:WEND
5200 IF dcp1/cp1>=0.1 THEN POKE bu+4,PEEK(bu+4) OR 8:
f$=LEFT$(aname$(p1),3)+" routs":GOSUB 3960:t=TIME:WHI
LE TIME<t+1000:WEND
5210 IF dcp2/cp2>=0.1 THEN POKE be+4,PEEK(be+4) OR 8:
f$=LEFT$(aname$(p2),3)+" routs":GOSUB 3960:t=TIME:WHI
LE TIME<t+1000:WEND
5220 RETURN
5230 LOCATE 2,2:PRINT"Making ready":ERASE ho:DIM ho(u
nits(p1)):REM HIDEOBST ***
5240 u1=0:b1=aa(p1)+2
5250 WHILE u1<units(p1):u1=u1+1:b1=b1+9
5260 IF PEEK(b1+1)=0 THEN 5330
5270 done=0:u2=0:b2=aa(p2)+2
5280 WHILE u2<units(p2) AND done=0:u2=u2+1:b2=b2+9
5290 IF PEEK(b2+1)=0 THEN 5320
5300 xm=PEEK(b1+2):xf=PEEK(b2+2):ym=PEEK(b1+1):yf=PEE
K(b2+1):dx=xm-xf:dy=ym-yf:ax=ABS(dx):ay=ABS(dy):IF ax
<2 AND ay<2 THEN done=1:ho(u1)=1:GOTO 5320
5310 GOSUB 4420:IF fb=0 THEN done=1:ho(u1)=1
5320 WEND
5330 WEND:RETURN
5340 bu=aa(p1)+15:FOR j=1 TO units(p1):REM CONCEAL **
*
5350 IF ho(j)=0 THEN POKE bu,PEEK(bu) OR 4:ELSE POKE
bu,PEEK(bu) AND 251
5360 bu=bu+9:NEXT:RETURN

```

Save the program under the file-name "NASEBY". Then load and save "MCODE" (see Chapter 3). This should be saved on to the same disk as "NASEBY" or immediately after it on tape. Finally, you need the data file which holds all the information about terrain and armies.

Readers who have typed in the wargaming utility programs should use these to build up the data file, as explained in the next section (Using the utilities).

Keyed-in method

Readers who wish to get the game up and running without further ado should proceed as follows.

First type in this *cipher program*:

```

10 REM *** NASEBYC ***
20 MEMORY 28823:CLS
30 PRINT"Storing army data"
40 FOR j=29997 TO 30589
50 READ x:POKE j,x
60 NEXT
70 PRINT"Storing map data"
80 READ x:POKE 30998,x:READ x:POKE 30999,x
90 j=31000:WHILE j<36670
100 READ x$:x=VAL("&"+"x$"):POKE j,(x AND 240)/16:POKE
j+1,x AND 15
110 READ x$:x$="&"+"x$":POKE j+2,VAL(x$)
120 j=j+3:WEND
130 PRINT"Storing cover table"
140 READ x:POKE 37000,x:READ x:POKE 37001,x
150 j=37002:WHILE j<37947:READ n:READ x
160 jj=j+n-1:FOR k=j TO jj:POKE k,x:NEXT
170 j=jj+1:WEND
180 PRINT"Storing movement table"
190 READ x:POKE 37947,x:READ x:POKE 37948,x
200 j=37949:WHILE j<38894:READ n:READ x
210 jj=j+n-1:FOR k=j TO jj:POKE k,x:NEXT
220 j=jj+1:WEND
230 PRINT"Storing CEV table"
240 FOR j=38894 TO 39091
250 READ x:POKE j,x
260 NEXT
270 SAVE "NASEBYD",B,29997,9095
280 PRINT "NASEBYD saved"
290 PRINT "Backup (Y/N)?"
300 a$="":WHILE a$<>"N" AND a$<>"Y"
310 a$=UPPER$(INKEY$):WEND
320 IF a$="Y" THEN 270
330 END

1000 DATA 179,152,3,0,2,58,117,137,118,0,0,0,0,70,65,73,82
1010 DATA 70,65,88,32,36,0,1,1,20,11,202,4,0,0,1,9,2
1020 DATA 18,8,83,4,0,0,250,9,2,18,9,235,4,0,0,250,7
1030 DATA 2,18,10,236,4,0,0,250,7,2,18,11,237,4,0,0,250
1040 DATA 7,2,18,12,238,4,0,0,250,7,2,19,9,239,4,0,0
1050 DATA 250,7,2,19,10,240,4,0,0,250,7,2,19,12,241,4,0
1060 DATA 0,250,7,4,17,7,255,4,0,0,50,8,4,17,13,255,4
1070 DATA 0,0,50,8,5,18,2,243,4,0,0,100,7,5,18,3,243
1080 DATA 4,0,0,100,7,5,18,4,244,4,0,0,100,7,5,18,5
1090 DATA 244,4,0,0,100,7,5,18,6,73,4,0,0,100,9,5,18
1100 DATA 7,252,4,0,0,100,8,5,19,3,245,4,0,0,100,7,5
1110 DATA 19,4,245,4,0,0,100,7,5,19,5,246,4,0,0,100,7
1120 DATA 5,19,6,246,4,0,0,100,7,5,19,7,247,4,0,0,100
1130 DATA 7,5,18,13,248,4,0,0,100,7,5,18,14,248,4,0,0
1140 DATA 100,7,5,18,15,249,4,0,0,100,7,5,19,15,249,4,0
1150 DATA 0,100,7,5,18,16,67,4,0,0,100,9,5,18,17,253,4
1160 DATA 0,0,100,8,5,19,13,250,4,0,0,100,7,5,19,14,250
1170 DATA 4,0,0,100,7,5,19,16,251,4,0,0,100,7,5,20,17
1180 DATA 242,4,0,0,100,7,6,12,1,254,5,0,0,110,8,6,14
1190 DATA 1,254,5,0,0,110,8,6,16,1,254,5,0,0,110,8,7
1200 DATA 28,13,35,4,0,0,33,7,67,72,65,82,76,69,39,83,26

```



```

1210 DATA 3,4,1,4,10,189,6,0,0,1,8,3,8,9,65,6,0
1220 DATA 0,200,9,3,8,10,235,6,0,0,200,8,3,6,9,236,6
1230 DATA 0,0,200,8,3,8,11,237,6,0,0,200,8,3,6,10,238
1240 DATA 6,0,0,200,8,3,8,12,239,6,0,0,200,8,3,6,11
1250 DATA 240,6,0,0,200,8,3,5,10,234,6,0,0,50,9,4,9
1260 DATA 5,255,6,0,0,33,8,4,9,16,255,6,0,0,33,8,5
1270 DATA 8,6,82,6,0,0,100,9,5,8,7,243,6,0,0,100,9
1280 DATA 5,8,8,244,6,0,0,100,8,5,7,7,245,6,0,0,100
1290 DATA 8,5,7,8,246,6,0,0,100,8,5,7,10,247,6,0,0
1300 DATA 100,8,5,8,13,76,6,0,0,100,9,5,8,14,248,6,0
1310 DATA 0,100,6,5,8,15,248,6,0,0,100,6,5,7,13,248,6
1320 DATA 0,0,100,6,5,7,14,248,6,0,0,100,6,5,7,15,248
1330 DATA 6,0,0,100,6,5,5,9,242,6,0,0,100,8,5,5,11
1340 DATA 242,6,0,0,100,9,7,3,6,35,6,0,0,0,0
1350 DATA 35,27
1360 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1370 DATA 09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,9E,D3,09,8F,09,8F
1380 DATA 09,8F,09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20
1390 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1400 DATA 09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,9E,D3,09,8F,09,8F
1410 DATA 09,8F,09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20
1420 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1430 DATA 09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,9E,D3,09,8F,09,8F
1440 DATA 09,8F,09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20
1450 DATA D9,A4,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1460 DATA D9,D5,09,8F,09,8F,09,8F,09,8F,09,8F,9E,D3,09,8F,09,8F
1470 DATA 09,8F,09,8F,09,8F,D9,D4,DD,20,DD,20,DD,20,DD,20,DD,20
1480 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1490 DATA DD,20,D9,D5,09,8F,09,8F,09,8F,09,8F,9E,D3,09,8F,09,8F
1500 DATA 09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1510 DATA D9,A4,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1520 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1530 DATA 09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1540 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1550 DATA DD,20,DD,20,D9,D5,09,8F,09,8F,09,8F,9E,D3,09,8F,09,8F
1560 DATA 09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1570 DATA D9,A4,DD,20,D7,53,DD,20,DD,20,DD,20,DD,20,DD,20
1580 DATA DD,20,DD,20,DD,20,D9,D5,09,8F,09,8F,9E,D3,09,8F,09,8F
1590 DATA 09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1600 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1610 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1620 DATA 09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1630 DATA D9,A4,DD,20,D7,75,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1640 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1650 DATA 09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1660 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1670 DATA DD,20,DD,20,DD,20,D9,D6,09,8F,09,8F,9E,D1,09,8F,09,8F
1680 DATA 09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1690 DATA DD,20,D9,A4,D7,6C,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1700 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1710 DATA 09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1720 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1730 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1740 DATA 95,E5,09,8F,95,5E,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1750 DATA DD,20,D9,A4,D7,62,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1760 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1770 DATA 09,8F,95,5E,09,8F,D5,E5,DD,20,DD,20,DD,20,DD,20,DD,20
1780 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1790 DATA DD,20,DD,20,DD,20,D9,D5,09,8F,09,8F,9E,D1,09,8F,09,8F
1800 DATA 09,8F,95,07,09,8F,D5,5E,DD,20,DD,20,DD,20,DD,20,DD,20
1810 DATA DD,20,D9,A4,D7,79,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1820 DATA DD,20,DD,20,D7,48,D7,69,D7,6C,9E,D1,09,8F,95,5E
1830 DATA 95,E5,09,8F,95,5E,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1840 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1850 DATA DD,20,DD,20,DD,20,DD,20,DD,20,D9,D5,9E,D1,09,8F,95,E5
1860 DATA 09,8F,95,E5,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1870 DATA DD,20,D9,A4,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1880 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20

```

```

1890 DATA 95,5E,09,8F,95,07,DD,20,DD,20,DD,20,DD,20,DD,20
1900 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1910 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DE,D1,D9,D5,09,8F
1920 DATA 09,8F,09,8F,95,5E,D9,D7,DD,20,DD,20,DD,20,DD,20,DD,20
1930 DATA DD,20,D9,A4,D7,4B,DD,20,0D,8F,DD,20,DD,20,DD,20,DD,20
1940 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DE,CC,0D,8F,D9,D5
1950 DATA 95,5E,95,07,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20
1960 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
1970 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DE,D3,DD,20,DD,20
1980 DATA 09,8F,09,8F,09,8F,09,8F,DD,20,DD,20,DD,20,DD,20,DD,20
1990 DATA DD,20,D9,A4,D7,65,DD,20,0D,8F,DD,20,DD,20,DD,20,DD,20
2000 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,0D,8F,DE,D3,DD,20,DD,20
2010 DATA D9,D5,09,8F,09,8F,09,8F,0D,8F,DD,20,DD,20,DD,20,DD,20
2020 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2030 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DE,D3,DD,20,DD,20
2040 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2050 DATA DD,20,D9,A4,D7,64,DD,20,0D,8F,DD,20,D7,42,D7,72,D7,6F
2060 DATA D7,61,D7,64,D7,6D,D7,6F,D7,6F,D7,72,DE,D3,DD,20,DD,20
2070 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2080 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2090 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DE,D3,DD,20,DD,20
2100 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2110 DATA DD,20,D9,A4,D7,67,0D,8F,0D,8F,DD,20,DD,20,DD,20,DD,20
2120 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DE,D3,DD,20,DD,20
2130 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2140 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2150 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DE,D3,DD,20,DD,20
2160 DATA DD,20,DD,20,DD,20,DD,20,D5,E5,DD,20,DD,20,DD,20,DD,20,DD,20
2170 DATA DD,20,D9,A4,D7,65,0D,8F,DD,20,DD,20,DD,20,DD,20,DD,20
2180 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DE,D3,DD,20,DD,20
2190 DATA DD,20,DD,20,D5,5E,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2200 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2210 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DE,D3,DD,20,DD,20
2220 DATA DD,20,DD,20,D5,5E,DD,20,D5,07,DD,20,DD,20,DD,20,DD,20
2230 DATA DD,20,D9,A4,D7,73,0D,8F,DD,20,DD,20,DD,20,DD,20,DD,20
2240 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DE,D3,DD,20,DD,20
2250 DATA DD,20,DD,20,DD,20,D5,07,DD,20,D5,E5,DD,20,DD,20,DD,20
2260 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2270 DATA D9,D6,D9,8F,D9,8F,D9,8F,DD,20,DD,20,DE,D3,DD,20,DD,20
2280 DATA DD,20,DD,20,DD,20,DD,20,D5,5E,DD,20,DD,20,DD,20,DD,20
2290 DATA DD,20,D9,A4,DD,20,0D,8F,DD,20,DD,20,DD,20,DD,20,DD,20
2300 DATA D9,8F,D9,8F,D9,8F,D9,8F,DD,20,DD,20,DE,D3,D9,D6,09,8F
2310 DATA D9,D7,DD,20,DD,20,D5,5E,DD,20,D5,E5,D5,5E,DD,20,DD,20,DD,20
2320 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,D9,D6,D9,8F
2330 DATA D9,8F,D9,8F,D9,8F,D9,8F,D9,8F,D9,8F,9E,D3,09,8F,09,8F
2340 DATA 09,8F,DD,20,DD,20,D5,E5,D5,07,DD,20,D5,07,DD,20,DD,20
2350 DATA D9,A4,0D,8F,0D,8F,0D,8F,0D,8F,DD,20,D9,8F,D9,8F,D9,8F
2360 DATA D9,8F,D9,8F,D9,8F,D9,8F,D9,8F,D9,8F,9E,D3,09,8F,09,8F
2370 DATA 09,8F,DD,20,DD,20,DD,20,D5,E5,D5,DD,20,DD,20,DD,20
2380 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,09,8F,09,8F,09,8F
2390 DATA 09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,9E,D3,09,8F,D9,D4
2400 DATA D5,5E,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2410 DATA D9,A4,DD,20,0D,8F,DD,20,DD,20,DD,20,D9,D5,09,8F,09,8F
2420 DATA 09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,9E,D3,09,8F,D5,07
2430 DATA 0D,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2440 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,0D,8F,0D,8F
2450 DATA 0D,8F,D9,D5,09,8F,09,8F,09,8F,09,8F,9E,D3,09,8F,D5,07
2460 DATA 0D,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2470 DATA D9,A4,DD,20,0D,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2480 DATA DD,20,DD,20,D9,D5,09,8F,09,8F,09,8F,9E,D3,09,8F,0D,8F
2490 DATA D5,E5,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2500 DATA DD,20,DD,20,0D,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2510 DATA DD,20,DD,20,DD,20,09,8F,09,8F,09,8F,9E,D3,09,8F,D5,5E
2520 DATA D5,5E,DD,20,D9,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2530 DATA DE,CD,DD,20,0D,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2540 DATA DD,20,DD,20,DD,20,09,8F,09,8F,09,8F,9E,D3,09,8F,D9,D7
2550 DATA 0D,8F,DD,20,D9,8F,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2560 DATA DD,20,DE,CD,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
2570 DATA DD,20,DD,20,DD,20,09,8F,09,8F,09,8F,9E,D3,09,8F,09,8F

```



```

3270 DATA 97,61,97,73,97,65,97,62,97,79,09,8F,9E,D3,09,8F,09,8F
3280 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
3290 DATA 00,8F,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,9E,CD,09,8F
3300 DATA 0B,8F,09,8F,09,8F,09,8F,09,8F,9E,CC,09,8F,09,8F,09,8F
3310 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
3320 DATA DD,20,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,0B,8F
3330 DATA 0B,8F,0B,8F,9B,8D,09,8F,9E,CC,09,8F,09,8F,09,8F,09,8F
3340 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
3350 DATA DD,20,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,9B,8A
3360 DATA 0B,8F,0B,8F,0B,8F,9E,CC,09,8F,09,8F,09,8F,09,8F,09,8F
3370 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
3380 DATA DD,20,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,0B,8F
3390 DATA 0B,8F,0B,8F,0B,8F,9E,CD,09,8F,09,8F,09,8F,09,8F,09,8F
3400 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
3410 DATA DD,20,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,0B,8F
3420 DATA 0B,8F,0B,8F,0B,8F,09,8F,9E,CD,09,8F,09,8F,09,8F,09,8F
3430 DATA DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20,DD,20
3440 DATA DD,20,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,09,8F,0B,8F
3450 DATA 0B,8F,0B,8F,0B,8F,09,8F,09,8F,9E,CD,09,8F,09,8F,09,8F
3460 DATA 35,27
3470 DATA 9,0,1,1,11,0,1,1,5,0,1,1,8,0,2,1,9,0,2,1,5,0,1,1
3480 DATA 9,0,2,1,8,0,1,1,6,0,1,1,10,0,7,1,2,0,1,1,6,0,1,1
3490 DATA 16,0,1,1,2,0,2,1,6,0,1,1,10,0,6,1,2,0,1,1,7,0,1,1
3500 DATA 9,0,2,1,4,0,5,1,6,0,1,1,10,0,3,1,2,0,5,1,6,0,1,1
3510 DATA 12,0,7,1,7,0,1,1,14,0,6,1,6,0,1,1,16,0,2,1,1,0,1,1
3520 DATA 6,0,1,1,17,0,3,1,6,0,1,1,26,0,1,1,18,0,2,1,6,0,1,1
3530 DATA 18,0,4,1,4,0,1,1,7,0,4,1,3,0,3,1,2,0,4,1,2,0,1,1
3540 DATA 5,0,4,1,2,0,7,1,2,0,4,1,2,0,1,1,5,0,5,1,5,0,3,1,8,0
3550 DATA 1,1,9,0,3,1,3,0,3,1,20,0,1,1,3,0,3,1,1,0,1,1,18,0
3560 DATA 1,1,3,0,5,1,12,0,7,1,7,0,1,1,11,0,2,1,13,0,1,1,5,0
3570 DATA 1,1,5,0,1,1,14,0,7,1,5,0,1,1,26,0,4,1,26,0,2,1,26,0
3580 DATA 1,1,26,0,1,1,26,0,1,1,26,0,1,1,26,0,2,1,7,0,1,1,18,0
3590 DATA 1,1,6,0,4,1,16,0,1,1,6,0,4,1,16,0,1,1,6,0,4,1,6,0
3600 DATA 35,27
3610 DATA 9,2,1,3,11,2,1,3,5,2,1,3,8,2,2,3,5,2,1,3,3,2,2,3,5,2
3620 DATA 1,3,3,2,1,3,5,2,2,3,8,2,1,3,6,2,1,3,10,2,7,3,2,2,1,3,
6,2,1,3
3630 DATA 6,2,1,3,9,2,1,3,2,2,2,3,6,2,1,3,10,2,6,3,2,2,1,3,2,2
3640 DATA 1,3,4,2,1,3,9,2,2,3,4,2,5,9,6,2,1,3,10,2,3,3,2,2,5,9
3650 DATA 2,2,1,3,3,2,1,3,12,2,3,3,4,9,7,2,1,3,3,2,1,3,10,2,2,
3,3,9
3660 DATA 1,3,6,2,1,3,16,2,2,3,1,2,1,3,1,2,1,3,4,2,1,3,9,2,1,3
3670 DATA 7,2,3,3,6,2,1,3,3,2,1,3,9,2,1,3,12,2,1,3,11,2,1,3
3680 DATA 6,2,2,9,6,2,1,3,1,2,1,3,16,2,4,9,4,2,1,3,7,2,4,3,3,2
3690 DATA 3,3,2,2,4,9,2,2,1,3,5,2,4,3,2,2,7,3,2,2,4,9,2,2,1,3
3700 DATA 5,2,5,3,5,2,1,3,2,9,8,2,1,3,9,2,3,3,2,1,3,2,9,3,2
3710 DATA 1,3,6,2,1,3,4,2,1,3,4,2,1,3,3,2,1,3,2,9,1,2,1,3,18,2
3720 DATA 1,3,3,2,5,3,4,2,1,3,7,2,7,3,7,2,1,3,11,2,2,3,13,2
3730 DATA 1,3,5,2,1,3,1,2,1,3,3,2,1,3,9,2,1,3,4,2,7,3,1,2,1,3
3740 DATA 3,2,1,3,23,2,1,3,2,2,4,3,26,2,2,3,11,2,1,3,14,2,1,3
3750 DATA 26,2,1,3,26,2,1,3,3,2,1,3,7,2,1,3,10,2,1,3,3,2,1,3
3760 DATA 19,2,1,3,6,2,2,3,7,2,1,9,18,2,1,3,6,2,4,9,3,2,1,3
3770 DATA 12,2,1,3,6,2,4,9,16,2,1,3,6,2,4,9,6,2
3770 DATA 28,7,0,0,0,0,0,0,0,0,0,0,30,30,30,41,41,15,0
3780 DATA 32,32,32,44,44,16,0,37,37,37,53,53,19,0,0,0,0
3790 DATA 0,0,0,0,0,0,0,0,0,0,0,0,31,31,31,45,45,16
3800 DATA 0,0,0,0,0,0,0,0,0,0,0,0,60,60,60,84,84,30,0,64,64
3810 DATA 64,90,90,32,0,73,73,73,107,107,34,0,43,43,53,63
3820 DATA 22,0,0,0,0,0,0,0,0,0,0,0,61,61,61,90,90,32,0,0
3830 DATA 0,0,0,0,0,0,54,54,90,56,65,32,0,85,85,140,92
3840 DATA 108,70,0,52,52,87,47,53,44,0,107,107,123,147,173,62,0
3850 DATA 54,54,107,73,73,54,0,48,48,80,44,49,40,0,0,0,0
3860 DATA 0,0,0,0,107,107,123,61,73,123,0,152,152,176,88,104,176
3870 DATA 0,127,127,147,73,87,147,0,147,147,213,107,127,213,0,
147,147
3880 DATA 213,107,127,213,0,127,127,147,73,87,147

```

When you have typed in this program, check it extremely carefully and then save it under the name "NASEBYC".

With disks Save NASEBYC on the same disk as the other programs. We use it to create a *data file*, called "NASEBYD". To create this file, all you have to do is to run NASEBYC. NASEBYC automatically creates the data file NASEBYD on the same disk as itself and the other programs (NASEBY and MCODE). NASEBYD is the data file which is loaded by the game program, NASEBY, when it runs.

With tapes The procedure is slightly different. Save NASEBYC on a separate tape. Now load NASEBYC. Then place the tape which has NASEBY and MCODE already on it, in the recorder, wound on to just beyond the end of the recording of MCODE. Now run NASEBYC. Press 'RECORD' and 'PLAY' and any keyboard key when asked to do so. The data file, NASEBYD, is then saved on the tape, immediately after MCODE. NASEBYD is the data file which is loaded by the game program, NASEBY, when it is run.

After you have used NASEBYC to create NASEBYD, you may never need to use it again. However, it is best to keep the recording in case you have made mistakes in entering the data. It may then be necessary to edit NASEBYC.

With NASEBY, MCODE and NASEBYD on the tape (in that order), or on the same disk, you are ready to play. Type RUN "NASEBY" and press ENTER. NASEBY is loaded and run; it then loads MCODE and NASEBYD, after which the game begins.

Using the utilities

For detailed instructions in using the utilities, see the earlier chapters concerned. Below we set out a suggested plan for putting together the data file for NASEBY.

1 RECRUITER: Prepare a file for each army, named "Fairfax's" and "Charles'" respectively, and save them under FAIRFAX and CHARLES. The data required is in Tables 7 and 8. Note that there are seven types of unit in *both* armies, but there are none of type 3 in Fairfax's army and none of types 2 and 6 in Charles' army. Starting positions are specified, and conform as closely as possible to the historical deployment of both sides at the commencement of the battle. If you prefer, you need not specify starting positions, in which event insert two deployment phases near the start of the game program, and type in the DEPLOY subroutine (lines 1300–1460).

There are five details. The values in the 'Status' column have the value '4' indicating that all units are hidden to begin with, plus the number which indicates the direction in which they are facing. The values in the 'Strength' column are one third of the actual strength of the units.

2 CARTOGRAPHER The terrain map has 27 columns and 35 rows (Figure 9.1). The Sulby Hedges are conveniently represented by '@' symbols, located in the lower half of the square. Similarly, it is better if the text characters on the map are in the lower half of the square. The colours in which we mapped the various types of terrain are: low ground = pastel green, high ground = yellow (appears olive green), tracks = white

(appears grey), Sulby Hedge symbols = yellow, woods = various tree symbols in black, Naseby town = pink, text letters = bright magenta. Save the finished map under the name "NASMAP"

3 **TABLER** Three tables are needed: NASEC for holding cover values for each square of the map; NASEM, for holding movement costs; NASCEV for the CEV values in close combat. In this game NASEC holds only zeros or '1's as there is no level 2 cover. The table consists of 27 columns and 35 rows, as does the map. The table begins at address 37000, the standard base address for a cover table. Using Figure 9.1 as a guide, enter '1' for squares in which the whole, or any part, of the area is a hedge or a slope. The slope squares are those which form the edges of the high ground. For example, squares 10/1 (column/row), 10/2, 11/2, 11/3, 12/3, 12/4, 13/4, 14/4 and 15/4 are slopes squares. Enter '0' for all other squares.

Table 9 NASCEV

Combat type	Attacking unit type	Attacked unit type						
		1	2	3	4	5	6	7
0:range=2	1	0	0	0	0	0	0	0
	2	0	30	30	30	41	41	15
	3	0	32	32	32	44	44	16
	4	0	37	37	37	53	53	19
	5	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0
	7	0	31	31	31	45	45	16
1:range=1	1	0	0	0	0	0	0	0
	2	0	60	60	60	84	84	30
	3	0	64	64	64	90	90	32
	4	0	73	73	73	107	107	34
	5	0	43	43	43	53	63	22
	6	0	0	0	0	0	0	0
	7	0	61	61	61	90	90	32
2:Mêlée (1st round)	1	0	0	0	0	0	0	0
	2	0	54	54	90	56	65	32
	3	0	85	85	140	92	108	70
	4	0	52	52	87	47	53	44
	5	0	107	107	123	147	173	62
	7	0	48	48	80	44	49	40
3:Mêlée (subsequent rounds)	1	0	0	0	0	0	0	0
	2	0	107	107	123	61	73	123
	3	0	152	152	176	88	104	176
	4	0	127	127	147	73	87	147
	5	0	147	147	213	107	127	213
	6	0	147	147	213	107	127	213
	7	0	127	127	147	73	87	147

NASEM also has 27 columns and 35 rows. It begins at address 37947. Using Figure 9.1, key 3 for any hedge or slope square and 9 for any wood square or square in Naseby village. Key 3 in any 25 squares of low or high ground to make these count as boggy or thorny squares. Scatter these squares irregularly over the map. They are not marked on the map and represent an invisible hazard. Key 2 for all other squares. For the CEV table, which has 7 rows and 28 columns and begins at 38894, use the values given in Table 9.

INTELLIGENCE Use this program to load and store the following files:

Army 1	FAIRFAX
Army 2	CHARLES
Map	NASEMAP
Table 1	NASEC
Table 2	NASEM
Table 3	NASCEV

The complete data file is saved under the name NASEBYD. With disks, it should be on the same disk as the NASEBY and MCODE files. With tape, it should follow after NASEBY and MCODE.

Notes for programmers

The sequence of the main program is:

```

20 Set screen colours.
30 Reserve memory; call START to initialise the game.
40 Calculate turn number.
50 Determine number of army to play this turn (phasing army).
60 Call PHASE for Advance Phase.
70 Call PHASE for Charge Phase.
80 Call PHASE for Battle Phase.
90 Call PHASE for Rout/Rally Phase for phasing army.
100 Calculate number of non-phasing army.
110 Call PHASE for Rout/Rally Phase for non-phasing army.
120 Set p1 and p2 to number of phasing and non-phasing armies.
130 Call HIDEOBST and CONCEAL to hide units of phasing army, ready
for next turn.
140 Call TURNEND; Save or continue?
150 To end game after move 24.
160 Back to line 40, if continuing.
170 Save game.
180-190 Restore mode 1 and its colours. End program.
```

Subroutines

The following subroutines appear for the first time in this program. Variables and arrays are defined in Appendix A.

UNITLINE/NASEBY A version of UNITLINE to display unit details as required for this game.

SYMBOLS/NASEBY The special symbols required for this game.

BRIEFING See Chapter 8.

BATTLE Resolves combat between two units at a range of 1 or 2 squares, or when opposing units are on the same square (*mêlée*).

HIDEOBST Runs through the units of army number *p1* to find out if it can be seen by any unit in army *p2*. The result is stored in an array *ho()*, where 0=visible and 1=hidden.

CONCEAL Runs through *ho()* and sets or resets the 'hidden' bit of the status byte according to whether the unit is hidden or visible.

The PHASE routine has been modified to disregard "D" and "M" keypresses, to respond to a "B" keypress, to display rosters of *both* armies, to omit the calls to MORTAR and DIGIN and to call BATTLE instead of FIRE, and to call BRIEFING.

10 Designing wargames

This book provides wargaming utilities and a selection of sample wargames that provide you with all that you require when designing your own computer wargames. There are three levels of game designing:

- 1 Extending or modifying the games from this book.
- 2 Adapting miniature or board wargames to run on the computer.
- 3 Inventing totally new computer wargames.

No programming knowledge is required for level 1, but a little may be required for level 2. The degree of programming ability required for level 3 depends on how innovative you are in conceiving new kinds of wargaming situations that are not covered by the subroutines described in this book.

Extending or modifying programs

The easiest type of modification is to use the same program, the same terrain and the same armies, with a new scenario. This gives you a new game without any need to alter the program or the data. As an example, here is a scenario for a new game that can be played with the JUNGLE ATTACK program. It requires the HIDECOVER subroutine to have been added to the program, as described in Chapter 7.

The period is the same as in the original scenario. The Australians have occupied most of the island and patrol the roads regularly. The only part of the island still under Japanese control lies to the north of the area shown in the map. It is reported that a Japanese transport plane has crashed in the jungle. The Japanese commandos on board have survived the crash and are attempting to make their way north to the Japanese occupied area. The aim of the Australians is to prevent them from escaping. In the initial deployment phases, the Australians deploy anywhere on the road or the surrounding open ground. The Japanese all deploy on a single square, which may be any jungle square in rows 9 to 21, at the option of the Japanese player. The aim of the Japanese is to move all units on to one of the track squares of row 1 (columns 10 to 15). On these squares, they are considered to have escaped into the jungle to the north, from where they

can easily reach Japanese-held territory. They are out of the game; they can neither attack the Australians nor be attacked by them. The Australians win if fewer than 10 Japanese units escape. The Japanese player has to decide whether to try to make a quick dash for the north, before the Australian reinforcements (deployed at either the eastern or western end of the road) arrive in turn 8, or to first reduce the strength of the Australian forces and also establish strong defensive positions ready to cover the escape of the majority of the Japanese. This decision has some bearing on the location selected for the 'crash' in initial deployment. The Australian player has to decide between searching the jungle with a limited number of men and perhaps missing the Japanese altogether, and digging-in to provide an interdictory barrage of fire along the road.

The first few times any wargame is played, it is probably better to use a simple set of rules, such as those given in the book. Later, when you have played the game several times, or when you have become a more experienced wargamer, you may wish to take more factors into account. In its original version or in the new version described above, JUNGLE ATTACK can be provided with numerous extensions involving little or no additional programming. The same applies to NASEBY and OMDURMAN. As described in Chapter 8, the BRIEFING routine gives players access to all stored data concerning the fighting units. This allows you to invent and implement a wide range of rules covering such aspects of the game as morale, arms carried, weapons skill, and disorganization. Extending the rules and discovering what effects this has on the outcome of combat is one of the more fascinating aspects of wargaming.

Adapting other wargames

Many of the points dealt with under this section apply also when you invent wargames of your own. Certain aspects of a game that originates as a miniature or board wargame will need special attention when the game is to be computerised. Usually, the greatest thought will have to be given to combat resolution, as will be explained later.

The features of the terrain map and the nature and composition of the armies are usually readily convertible to the computer, using the CARTOGRAPHER and RECRUITER utilities. The TABLER utility is used for storing data relating to cover and movement. It may be necessary to modify certain lines in the ADVANCE subroutines. The way this routine from JUNGLE ATTACK is modified for use in NASEBY and OMDURMAN shows how this may be done. In OMDURMAN, movement allowances do not affect the placing of unit symbols on the map since the map represents an operations table in this game. Thus the version of ADVANCE used in OMDURMAN is suitable for those games in which movement rules are complicated and you need to be able to move units without any restrictions.

Another subroutine that will need amending is SYMBOLS. Up to 22 special symbols may be designed, as described in the User Instructions. These symbols may be used to represent the same types of units on both

sides, since each army has its own distinctive pair of colours for symbols. If more than 22 symbols are needed, you may employ any of the Amstrad's standard set of graphics characters, including those associated with control codes (see Chapter 2).

As explained in Chapter 4, resolution of combat in miniature wargaming and board wargaming generally depends on combat resolution tables. Random factors are introduced by relying on the throw of a die or dice. Close examination of combat resolution tables shows that a significant number of them are not soundly based even though they may be convenient for their intended purpose. One might as well replace such tables by one of the routines described in Chapter 4. A little adjustment of the standard deviations or of the values in the CEV tables of the routines used in this book will often provide an acceptable and possibly superior substitute. However, many combat resolution tables are the result of painstaking research and it is difficult to improve on them. Again, the routines used in this book can probably be 'tuned' to produce the same results. The best approach is to key in the routines separately from the wargame program. Use the combat resolution table several times (say, 50 times) in the way prescribed by its designers and note the results. Then run the combat resolution routines as many times and note the results. The two sets of results should be equivalent. If not, adjust the values in the computer subroutines accordingly.

You may prefer to use the published combat resolution table as it stands, and restrict the computer to performing the dice-throwing. This approach is probably the best when the tables you wish to use differ considerably in their basis from the routines used in this book. Below we describe three functions and two subroutines that automatically 'throw dice'. The five types of dice-throwing most frequently used by wargamers are:

- 1 Single normal die: the normal die has faces marked 1 to 6. Values 1 to 6 are equally likely to occur.
- 2 Pair of normal dice: values 2 to 12 are obtainable, but values around 7 are most likely to occur.
- 3 Single average die: This has its faces marked 2,3,3,4,4, and 5. Values 2 to 5 are obtainable, but values 3 and 4 are twice as likely to occur as 2 and 5.
- 4 Pair of average dice: values 4 to 10 are obtainable, but values close to 7 are more frequently obtained than with a pair of normal dice.
- 5 Percentage dice. These are usually polyhedra with 10 faces, marked 0 to 9. They are differently coloured; one gives the 'units' digit and the other the 'tens' digit. Values between 0 and 99 are obtainable, and any value in this range is equally likely to occur. In some games, a throw of two zeros is taken to represent 100, so the range is from 1 to 100.

Function NDIE This function simulates a single normal die, returning a value between 1 and 6.

```
10 DEF FNdie=INT(RND(1)*6)+1
```

Function NDICE This simulates the throw of two dice and returns their total.

```
15 DEF FNdice=FNndie+FNndie
```

Subroutine AVDIE This produces the same result as the throw of one average die. The average die is often used to simulate the more reliable behaviour or responses of veteran troops, as opposed to novices.

```
50 avdi=INT(RND(1)*6)+1:IF avdi=1 THEN avdi=3:REM AVD
I ***
60 IF avdi=6 THEN avdi=4
70 RETURN
```

Subroutine AVDICE This produces the same result as the throw of two average dice. A pair of such dice is also used to simulate behaviour or responses of veteran troops.

```
60 avdi=INT(RND(1)*6)+1:IF avdi=1 THEN avdi=3:REM AVD
ICE ***
70 IF avdi=6 THEN avdi=4
80 avdi2=INT(RND(1)*6)+1:IF avdi2=1 THEN avdi2=3
90 IF avdi2=6 THEN avdi2=4
100 avdi=avdi+avdi2
110 RETURN
```

Function PDIE This is similar to NDIE, except that it produces values in the range 1 to 100. If you wish to obtain values in the range 0 to 99, delete the '+1' from the end of the definition line.

```
10 DEF FNpdie=INT(RND(1)*100)+1
```

The dice functions can be added to the START subroutine, and the dice subroutines can be appended to the program. These functions or subroutines can be called at any stage of the program. If a dice function is to be used in conjunction with advancing a unit, a line such as this could be inserted in the ADVANCE routine:

```
100 LOCATE 10,12:PRINT FNdie
```

This is how to call a dice subroutine:

```
200 LOCATE 10,12:GOSUB 8500:PRINT avdi:REM AVDICE
```

As soon as the ADVANCE routine begins, the random value is displayed on the screen. It can then be used as required when moving the unit. Any other of the dice functions or subroutines can be substituted in the lines above. The functions or subroutines can also be called in combat or firing routines by inserting these calls in FIRE or BATTLE subroutines.

The wargaming magazines and several of the books listed in Appendix B provide detailed suggestions for wargames that can easily be adapted to the computer. They may also provide accounts of historical battles which you can turn into wargames, as described in the next section.

Inventing wargames

There are four fields that the wargame inventor may wish to explore:

1 *Period wargames.* Wargames set in a particular period, but not related to any specific incident. JUNGLE ATTACK is an example. Such games generally tend to be based on a particular tactical situation which the players are required to resolve.

2 *Simulations of historical battles.* Wargamers may prefer to follow through the exact course of the battle. Others prefer to begin as the real battle began but, from then on, to try their own tactics and, maybe, alter the course of history. Military magazines and books often provide all the details necessary to the game designer.

3 *Future battles.* A future war that is the subject of several commercial programs is World War III which, curiously, always seems to be centred in Europe. Future battles may be set well into the next century and beyond, and fought with imaginary weapons of great power. Future wargames may also be fought in deep space, on other planets or even in other galaxies. The design of a future battle wargame relies heavily on the creative powers of the inventor. It is probably more difficult to write a coherent and convincing future wargame than to write any other kind.

4 *Fantasy wargaming.* This has recently become a cult in certain sections of the wargaming public but is scorned by others. Not only is the designer free from constraint in the invention of locations and weapons, but also in the creation of the combatant beings. Spells and magic combined with ingenious weaponry give fantasy wargaming a flavour all its own. As with future wargaming, appreciable talent is required to produce a convincing game. The aspiring fantasy wargamer might well begin by basing games on the worlds created by well-known fantasy writers, such as Tolkien.

Wargaming system

Before suggesting how to set about designing a wargame, we will outline the main features of the system used in this book. Figure 6.5 shows the structure of a typical wargame program, while Figure 5.1 shows how memory is allocated for the storage of machine code and wargaming data. For fuller details on data storage refer to Chapters 2, 3 and 5. Most of the

initialising of the game is performed by the subroutine *START*, which is called in the first few lines of the program. Among other things, it assigns values to certain variables that are used frequently in the game. Although they are referred to as variables, it is more appropriate to refer to them as constants, since they are not assigned other values during the game. As well as those assigned by *START*, which are required in almost every game, there are other fixed variables which are assigned values at the beginning of the main program for that particular game. The fixed variables include:

Map constants: *map*, *nrows*, *ncols*, *cmap*, *amap*, *exs*, *lwr*, *rgt*.

Army detail arrays and constants: *aa()*, *units()*, *pe()*, *pa()*, *aname\$()*, *maxu*.

Combat resolution constants: *bcev*, *cevr*, *cevf*.

Other constants: *gturn*.

Variables which hold their value for one player-turn: *turn*, *numarmy*.

The subroutines expect that the variables in the first four categories above will never have their values altered during the game. Similarly, those in the last category will not be altered during the turn.

Most lines in the main program are simply calls to various subroutines such as *DEPLOY*, *PHASE*, *CLOSE*, *HIDEOBST* and *TURNEND*. The order in which these are called and the number of times each is called determine the pattern of play. In most programs the majority of the calls are part of a program loop. The loop begins with a line to calculate the turn number. The variable *turn* is the total number of player-turns reached at a given stage of the game. This number is taken from memory, incremented and stored again in memory. This stored value is saved along with other data when a game is saved. This allows the game to be resumed at the stage following that at which it was saved. The turn number, the number of *game*-turns, as displayed on the screen, is not necessarily the same as *turn*. In most wargames, in which there are two armies, each game-turn consists of two identical player-turns. We go round the game loop twice for each game-turn. In some games the two sides might need to go through a different sequence of phases. In such games, the two sequences would be programmed one after the other in the game loop. We would go round the game loop once for each game-turn.

The game loop may contain conditional statements to vary the sequence or to introduce special phases at different stages of the game. For example, additional Australian units are to be deployed at turn 8 in *JUNGLE ATTACK*.

The game loop ends with a call to subroutine *TURNEND*. This displays the current status of the units of the phasing player and gives the option to save the game or continue, unless the game is ended because the full number of turns has been played.

Background research

If you are designing a game based on an historical battle, or set in a particular period, your first step in designing the game will be to research

the background. What were the events leading up to the battle? What were the stages of the battle itself? Was it a victory for one side or was it indecisive? Who were the most important commanders? What were the tactics employed? What was the composition of the armies? How high was their morale? How well were they trained? What were the special features of each type of unit? What were their arms and how effective were they? What was their armour, if any? Did the terrain provide cover? Did the terrain offer special advantages or difficulties? Were these the same for both sides? Were there any special features that played a significant part in the outcome of the battle, such as the weather, shortages of ammunition, smoke from gunfire, attrition due to hunger, or rebellious troops? Attention to such special features and working them into the fabric of the game will add interest and distinction to the game.

If you are inventing a future or fantasy wargame, many of the same questions will need to be asked before you begin to design the game. An imaginative mind compensates for the relative lack of reference books.

Design procedure

The first design points to consider are the *scales* of the game. Choice of scales has been discussed in Chapter 3. Once these are established you can start to work on the *map* of the terrain. In an historical game, this should cover a rather larger area than that over which the original battle was fought. This allows scope for action extending in other directions than in the original battle. If the scenario is such that the main action is likely to be centred on a particular region of the map, make sure that this area is well centred on the screen. The screen displays columns 1 to 20 and then, when the section to the east is displayed, it shows columns 11 to 31. It is best, therefore if any key feature on the map is located in the region of columns 9–11 or columns 19–21. A vital feature located in column 17, for example may be inconveniently too far to the east on one screen display, and too far to the west on another. Similar considerations apply to the placing of key features with regard to rows.

The next step is to write out the *army lists*. In basing an army list on the known forces taking part in an historical battle, it may be necessary to compromise between accuracy and playability. In particular, it is important that there should be a suitable number of units on each side. If units are too few, play is less flexible, there are fewer ways the units can be deployed. On the other hand, if the number of units is too great, the map becomes cluttered with symbols and each turn takes too long to play. We have found that a convenient number of units is between 15 and 35 units on each side. It is not essential for the number of units on each side to be equal. In NASEBY, for example, the Royalists have fewer units, and their infantry units represent fewer men than those of the Parliamentarians. This is offset to a certain extent by the Royalists being well-trained veterans (taken into account in the CEV table) and having higher initial morale (which makes them less likely to desert). In other games the armies may be very unequal, in accord with historical fact. As explained later, this does not necessarily

give the player on the stronger side a better chance of winning the game. When writing the army list, decide which special features are important for the game, and allocate a 'detail' for each. Morale has been of crucial importance in many battles. Large-scale desertions on one side have led to its defeat. If you are wargaming such a battle, morale must obviously be one of the details and must be incorporated into the rules. In other battles, shortage of ammunition has played a vital part. In wargaming such a battle, allocate a detail for 'rounds of ammunition' and devise the programs so that units can no longer fire when ammunition has run out. The selection of factors that decide the combat effectiveness of the units, gives each game its own distinctive appeal. The mechanism for *resolving combat* depends mainly on the unit scale. In a skirmish game, in which one unit represents one or very few warriors, it is best to simulate the fire of individual weapons, as was done in JUNGLE ATTACK. Close combat is resolved by routines similar to CLOSE, with its CEV table. In battle games, we 'average out' the results of many soldiers firing their weapons at multiple targets and express the results in a number of CEV tables, as in NASEBY and OMDURMAN. In writing your own games, you may find that you can use the same tables as we have used in this book. At least, these will be sufficient to get the game started. Later, as you test-play the games, certain values such as the standard deviations, or the values in the CEV tables may be adjusted to 'tune' the game to your requirements. There usually has to be a compromise between realism and playability.

At this stage of the design of the game, you will probably be formulating the *rules*. Reference to existing games or to published rule books will help. You will probably have to invent additional rules to cater for the special situations occurring in the game. It is important that the rules should reflect the style of combat and tactics that the commanders used at the time the battle was fought. Once again, it is necessary to play-test the game before finally drawing up the rule-sheet. In testing the games in this book we found that the rules we began with did not cover all possible situations and sometimes led to a lack of realism. Of course it is not worthwhile trying to provide rules for every conceivable circumstance, even if it were possible! In such a complex activity as a wargame, there must be rare configurations of terrain and military forces which were not foreseen by the game designers. In such situations, the players should apply common sense.

You will also need to state clearly what each side is to achieve during the game. These should preferably be as clear-cut as possible. You will also need to define under which conditions victory is to be accorded to one side or the other. As with many wars, there may be no victor, and the conditions set for victory should allow for an indecisive result. In wargaming, there is so much of interest at each stage of the battle, that winning or losing is, to many people, a minor consideration. However, it is essential to state victory conditions so that players may know when the game is over! Incidentally, it is best to be generous when fixing the maximum number of turns in the game. There is nothing worse than having spent half the game in planning a Grand Tactic and, just as one is about to close the trap on the enemy to have the screen announce "Game finished". It is not essential that

a player shall have to achieve a military victory in order to win the game. There have been many occasions in history when a battle has been lost but, nevertheless it has been a tactical or strategic victory for the loser of the battle. A battle may be fought to gain time, to delay the advance of an enemy while some other manoeuvre is completed, or to await the arrival of reinforcements. Thus the victory conditions may not require the player to annihilate or repel a more powerful enemy but merely to stem its advance for a prescribed number of turns. Thus the armies may be unequal, but the tasks set the players may be such that each player has an equal chance of winning the game. This leads us to the final point. Unless the players are interested strictly in re-creating history, a wargame must be fair to both players. The combination of initial deployment, balance of the armies, rules and victory conditions must be such as to result in a game that is instructive, enjoyable and just.

Independent action 11

When a battle is in progress and a commander wishes to send a unit on a mission, an order has to be given to that unit. There may be a considerable interval between the writing of the order by the commander and its delivery to the officer in charge of the unit. The length of time varies according to the method used for transmitting orders. Methods used today or in the past include field telephones, radio sets, runners, dispatch riders on motor cycles, gallopers, semaphore, heliograph and smoke signals. Each of these methods has its advantages and disadvantages. Radio is fast, but radio sets are easily damaged and may not work properly under battle conditions. They can be jammed by the enemy, the messages can be received by the enemy and may possibly be deciphered. The transmission of radio signals may give away the location of the unit sending the signal. It may be essential to maintain radio silence at a critical stage of an attack. Telephone lines need to be set up, and are liable to damage under battle conditions. Semaphore and heliographs require line-of-sight visibility, and may be obscured by smoke. Heliographs depend on the sun shining. Runners and messengers are slow, and may be killed or captured. The dispatches may easily fail to arrive or be delivered into the wrong hands.

The factors above also apply to reports being sent back to the commander from units in the field. Unless the unit is in full view of the commander, there is no way in which the commander can be sure exactly where it is or exactly what it is doing. If the unit is far away, an order that has just been sent and is on its way to the unit, may be totally inappropriate by the time it arrives. Conversely, by the time a report arrives from a unit, it may be too late for appropriate action to be taken. The commander has to function with a double and uncertain time-delay between himself and the units under his command.

The result of this situation is what has been termed the 'Fog of War'. This has proved to be a decisive factor in many battles and campaigns, yet it is a feature that is lacking from most wargames. In JUNGLE ATTACK and NASEBY, we look at the screen and see where all our units are *now*. When we move a unit or engage to in combat, this has *immediate* effect. The same situation occurs in miniature wargaming and in board wargaming, with certain exceptions. In the version of wargaming known as *Kriegspiel* and in

miniature wargaming based on that system, there is a referee who is the only person to see the map on which the tokens representing the units are placed. The referee accepts written orders from the players and, a few turns later, moves the tokens accordingly. The delay in putting the orders into effect is judged by the umpire to be that which would be required for the transmission of the orders to the unit concerned. Similarly, reports on the activities of the units are given to the players after an interval of a few turns. The players are thus subject to the Fog of War, and the game becomes at once more realistic (and more challenging!) to play.

Mission

This is a subroutine by means of which the computer takes over many of the functions of an umpire. We have incorporated this subroutine in the game presented in the next chapter, OMDURMAN. A game such as this is an experience totally different from a normal wargame. One unit in each army represents the army headquarters. The HQ unit of your side is where you, the commander, are located. All actions of your army are directed as if you are at that location. The terrain map shows where the HQ unit is located and also the locations of any units that would be visible from HQ. You are able to move the HQ unit during the game, just like any other unit, by giving it orders, as explained later.

First of all, let us explain how it is decided which units are visible from HQ. The desert around Omdurman has an undulating surface which makes it impossible to see units more than about 2 km distant. In this game, there is a hidden movement routine called HIDEEDIST. It hides all units that exceed a given distance from the HQ unit. Most units may be close to HQ, and therefore visible, at the beginning of the game, but they disappear out of sight when they are sent off on a mission. From then on, the commander knows where they are only if they send back a report as to their position, or their position is reported to HQ by other units. Reports are delayed for several turns, depending on how long it would take a galloper to travel from the reporting unit to HQ. Occasionally, a report is 'lost' on its journey, so the information never reaches HQ. Until the position of a unit has been updated by a report received by HQ, the map displays the symbol of the unit in its *last known* position.

Operations table

Owing to the delay in receiving reports from outlying units, the screen display does not convey the same information as it does in the other games. In OMDURMAN, it is in the nature of an Operations Table. Most readers are familiar with films of World War II in which there are scenes of the Operations Room. There is a large table or wall-map on which numerous operators move tokens to represent the latest known positions of the fighting units of both sides. The operators are supplied with information as it arrives at HQ and continually update the display. The commander sits in

a gallery overlooking the table, trying to penetrate the Fog of War and to interpret the information presented by the Operations Table.

Although the commanders at the Battle of Omdurman obviously did not have Operations Rooms, they must both have carried in their minds a map of the battle field, updated by what they could see with their eyes and by news brought by messengers from distant units. It is this map that the subroutine OPSTABLE represents on the screen.

It is important to be able to distinguish between what the commander *knows* is actually happening, because it is in sight, and what he *believes*, on the basis of the delayed information reported back by dispatches. To do this we display the known positions of HQ and units in sight of HQ in the normal way, but the most recently reported positions of other units are displayed in reverse. That is to say, if the symbols are normally displayed in white on black, the reverse symbols are displayed in black on white.

Issuing orders

When the MISSION routine commences, the turn number and name of the army are displayed. The map shows the actual or reported positions of units, as described above. The cursor control keys are not used for moving or firing units in games which use MISSION, in fact, the cursor is in the bottom right hand part of the screen and cannot be moved onto the map. The use of the cursor keys is restricted to changing the section of map being displayed. Units are controlled by *ordering* them to move or fight as described shortly. Even the HQ unit has to be sent orders. You change map sections in the usual way. When you have examined the map, and wish to issue orders, move the cursor to 'F' and press the space bar. Then a message 'UNIT?' appears at the bottom of the screen. This invites the commander to begin issuing orders. Key in the number of the unit to which the order is to be sent. For convenience, the unit symbols in OMDURMAN show simply the unit number, instead of a pictorial design. The next message is 'ORDER?', unless there are five orders already on their way to the unit, in which case the message 'No couriers' is displayed. There must obviously be a limit to the number of soldiers that can be spared to scurry around the battlefield! If this message appears you may have to wait a few turns before sending that unit an order, while some of the orders in transit are cleared. Assuming that the message 'Order?' appears, you can then send an order to that unit. To save lengthy typing, orders are sent by pressing the numeric keys:

Key	Order
1	Halt
2	Attack/Reinforce target unit
3	Engage enemy on sight
4	Report contacts
5	Report combat
6	Report arrival
7	Acknowledge message
8	End of message
9	Cancel

Some further explanation of these messages is required. Any of the messages 2 to 8 order the unit to advance toward a target, which may be a particular square or a particular unit of either army. You are asked to specify the target when all orders to the unit have been issued (see later). The special meanings of each order are as follows:

Halt This causes the unit to stop whatever it is doing and wait for further orders. A halt message comes into effect immediately it is received by the unit. This may be several turns after the message was sent, if the unit is far from HQ. 'Halt' is different in effect from the messages 2 to 7, because it is always sent as a single instruction. If you wish a unit to stop what it is doing immediately and then start to do something else, first send a 'Halt' message, followed by a second message to give it its new instructions. The second message can be sent in the same turn as the halt message. If you send a new order that is *not* preceded by a 'Halt' message, the unit carries on obeying the existing order until the task is completed, *then* begins to obey the next one.

Attack/Reinforce target unit The *target unit* may be a particular unit of either army, or it may be any unit located on a particular square (assuming that the square is still occupied when the unit reaches it). The effect of this message depends on which army the target unit belongs to. If the target unit is a friendly unit, this order instructs the unit to advance to the target unit and then to reinforce it. A single unit is formed, automatically, with strength equal to the total strength of the two units. If the target unit belongs to the enemy army, the effect of the order is to make it advance to the target unit and then engage it in *mêlée*. If a unit is given one of the orders 3–7, but *not* 2, it advances to a square adjacent to the target unit or square but does not enter the square. The attack/reinforce order is particularly useful for finding units which have been out of touch with HQ for a long time. As the searching unit proceeds on its journey, it scans the surrounding area, and will 'find' the target unit provided that it passes within five squares of it. Once it finds the unit, it homes on to it, to reinforce it or engage it in combat.

Engage enemy on sight After it has completed each move, the unit engages in combat with any enemy unit it finds on adjacent squares to the left and right. Combat is at the range of one square, not *mêlée*.

Report contacts If, at the end of its move, there is a friendly unit or enemy unit on an adjacent square to the left or right, this fact is reported to HQ, with full up-to-date details of the friendly or enemy unit. Details include its present location, and strength. These details will probably be somewhat out-of-date by the time the report reaches HQ, but these represent the 'latest news from the front' and it is with these details that the Operations Table display is updated. In ordering units to report back, remember that there is a limit of five reports in transit between each unit and HQ at any one time. If units are given an excessive number of 'report' instructions it may happen that there are not enough 'couriers' available. Valuable information may be lost, while trivial information gets through to HQ. It requires judgement on the part of the commander to obtain the maximum amount of useful information without overloading the system.

Report combat The unit sends reports each turn that it engages in combat. This report includes full details of the state of both units after combat has occurred. Some of the remarks in the previous paragraph apply here too. An additional feature is that the routine also causes the enemy (attacked) unit to send a report back to its own HQ, giving details of its own status and that of the attacker.

Report arrival The unit reports back when it has arrived at its target square, or the target unit.

Acknowledge message When the unit receives this message, it sends back an acknowledgement, and informs HQ of its present state and location. This is a useful way of obtaining information on the whereabouts of the unit, but take care not to ask for information too frequently, or the channels of communication will become clogged.

End of message You may key in as many of the messages 2 to 7 as you wish. These will be combined and transmitted as one order. When you have finished, key '8' to indicate the end of the message. An order which is only 'end of message' without other orders, is taken to mean "Advance to target".

Cancel If you make a mistake when issuing an order, key '9'. Then issue the order again.

If you have issued a 'Halt' order (1) you do not have to key '8' and you are not asked to indicate a target.

If you have issued orders 2 to 7 and after you have keyed '8', to indicate 'End of message', the screen displays 'Column?'. If you intend the unit to advance to a particular target square, key the column number of that square. You are then asked 'Row', to which you respond by keying the row number of the target square. If, instead of selecting a target *square* you prefer to instruct the unit to advance toward a target *unit*, key '0' in response to the 'Column?' query. You will then be asked to key in the number of the army and the number of the unit that is to be targeted.

This completes the issuing of the order and the message 'Unit?' appears again. This invites you to issue another order which may be to the same unit, or to a different unit. The procedure above is then repeated. You may issue as many orders each turn to as many units as you like (but not more than five to each unit). When you have finished, key '0' in response to 'Unit?'. The message 'Accepted' is displayed to indicate that this stage of the turn is ended.

A delay follows in which the orders are processed, being 'sent on their journey' to the units concerned. A delay time is calculated for each order taking account of the distances between a unit and HQ at the time the order was issued. This is approximately one turn for every three squares between sender and receiver. Thus, units within three squares of HQ, including the HQ unit itself, receive their orders and respond to them in the same turn. During the processing an order may occasionally be 'lost', at random, so will never arrive at its destination. You are not told when an order is lost so, if a unit seems to be a long time in responding or is apparently ignoring an order, send the order again in a subsequent turn.

Viewing the ops table

When issuing orders you will probably wish to consult various sections of the map. To do this, press the ESC ('Escape') key *twice*. This interrupts the MISSION routine to allow you to change map sections. Use the cursor control keys in the normal way to change sections. When you have obtained the section you require, move the cursor to 'F' and press the space bar. This allows the MISSION routine to continue from the stage it was at before you interrupted it.

You can use ESC to change map at any stage during the issuing of orders, or later, while units are being advanced or when reports are being received. Note that changing map sections interrupts the processing of orders and reports, which is not resumed unless you move the cursor to 'F' and press the space bar.

Advancing

The next message to appear is 'Advancing', indicating that units are continuing to obey orders previously received or are starting to obey new orders that have 'arrived' that turn. Units move toward their target square, avoiding terrain (such as rivers) on which they are not allowed to move. They also avoid squares occupied by other units (friendly or enemy) except when ordered to attack/reinforce the target.

Movement points are taken into account and normally the full movement allowance is used. The algorithm for advancing the units has to be a relatively simple one, so that the time spent in this stage is not excessive. In spite of this the result is very realistic. However, as in a game in which the players are directly responsible for movement, situations may arise in which a unit is unable to make a legal move. For example, it may be surrounded by squares onto which it is not allowed to move, either because they are prohibited terrain (such as a river square) or are already occupied by other units. In such an event, the unit makes two attempts to find a legal move and then abandons the move until the next turn.

As soon as all units have completed their moves, the message "All advanced" is displayed.

Reports

The computer then displays all reports that have arrived at HQ during the turn. The first display for each unit tells you which unit is reporting, where it was located, when it sent the report, and during which turn the report was sent. To save space, this information is presented in a compact form. You will see a message such as '12 at 3/19 T5'. This is interpreted as 'Message from unit 12, located at column 3, row 19, sent in turn 5'. On the line below you are shown the usual unit details: its symbol and strength (as it was at the time of sending the report). As soon as you have read (and

perhaps noted down) this information, press the space bar to receive the remainder of the message. The messages may be:

'Acknowledged' Acknowledging the receipt of an order, if requested to do so.

'Arrived' Arrived at the target square or on the square adjacent to a target unit. If the unit has been ordered to attack the target, it will now be in *mêlée*. If it has been ordered to reinforce the target, its own strength will be reported as zero. The strength of the reinforced unit is reported, if it has been ordered to report contacts.

'In combat' The line below this message displays the details of the enemy unit with which it is in combat. The strength will be that *after* the resolution of combat. This message appears for a unit in your own army only if you have ordered it to report combat. But, if a unit in your army is attacked by an enemy unit, you will receive this report, even though you have not requested it.

'In contact' The line below displays the details of the contacted unit, friendly or enemy. Remember that, at any stage while receiving reports, you can press ESC to change map section. The units will be displayed in their new positions.

When you have read the display, press the space bar. The display now reads 'End of message' and the next report begins. There may be more than one report from a unit, after which reports from other units are displayed.

Paper-work

From the description above, it is obvious that it is extremely useful, if not essential, to have a note-pad beside you when playing a game involving independent action. Although the map display shows the locations of the units as most recently reported, it does not show the strength of each unit. In addition you need to keep a record of orders issued, so that you will know what messages to expect from the units. Playing a game of this type emphasises the fact that successful generalship is less a matter of gallantry and quick thinking on the battlefield, than of painstaking desk-work back at HQ.

We have found that, with a little experience, it is easy to jot down brief notes of orders issued and of reports received. As in actual warfare, the extent to which you can do this and so help dispel the Fog of War, determines your chance of victory.

Notes for programmers

The main subroutine, MISSION, is given in lines 5370 to 7040 in the listing of Chapter 12. It comprises four stages, the commander issues orders (5370 to 5660), orders are transmitted to the units (5670 to 5910), the units advance, attack, and make reports (5920 to 6760), reports are transmitted to HQ and displayed to the commander (6770 to 7040).

There are three associated subroutines: STACKS is used at the beginning of the program to prepare the data stacks (see below); REPORT, prepares reports from units; COMBAT resolves combat, using the previously-described subroutine RESCOM. There are certain limitations in using MISSION. One is that there can be only two armies, not three or four. The number of units in the two armies must not exceed 33. In OMDURMAN, we have 16 units in one army and 17 in the other. This is hardly a limitation, for you will probably find that 16 or 17 units are enough to cope with in a game of this nature! The routine assumes that only one unit is allowed to occupy each square. The routine makes use of a movement allowance table, but not a cover table. Combat resolution requires the use of a CEV table (see Chapter 4) consisting of two sub-tables, one for combat at one-square range, the other for mêlée.

Data storage

Data relating to the army units is all stored in the memory space allocated to army details. It is in six blocks:

<i>Base address</i>	<i>Contents</i>
aa(1)	Up-to-date details of Army 1
aa(2)	Up-to-date details of Army 2
aa(3)	Details of Army 1 as reported to Army 1 HQ
aa(4)	Details of Army 2 as reported to Army 2 HQ
aa(5)	Details of Army 1 as reported to Army 2 HQ
aa(6)	Details of Army 2 as reported to Army 1 HQ

Thus, for Army 1, the routine uses details stored in *aa(1)* when moving a unit and when engaging it in combat. These details are also transferred to reports. They are up-to-date at that time, but become out of date to a greater or lesser extent before the report reaches HQ. When reports on units in Army 1 reach HQ, they are transferred to memory at *aa(3)*. Reported details of enemy units (Army 2) are stored in memory at *aa(6)*. The information in *aa(3)* and *aa(6)* is used for displaying reports, and for showing the position of units on the map (except for units that are within sight of HQ). The data in *aa(2)*, *aa(4)*, and *aa(5)* are used in a similar way for providing information to the commander of Army 2.

Orders are stored in the Order Stacks, one for each unit. Each stack requires 24 bytes, allocated as follows (*bos* is the base address of the stack):

<i>bos</i>	Numbers of orders on the stack (1-5)
<i>bos</i> +1 to <i>bos</i> +5	Delay for each order (turns)
<i>bos</i> +6	Order byte
<i>bos</i> +7	Target row
<i>bos</i> +8	Target column
<i>bos</i> +9 to <i>bos</i> +11	} 1st order
<i>bos</i> +12 to <i>bos</i> +14	
<i>bos</i> +15 to <i>bos</i> +17	
<i>bos</i> +18 to <i>bos</i> +20	2nd to 5th orders
<i>bos</i> +21 to <i>bos</i> +23	Executing order

If the unit is targeted on a *friendly* unit instead of a square, $\text{bos}+7$ holds the unit number and $\text{bos}+8$ is zero. If the unit is targeted on an *enemy* unit, $\text{bos}+7$ holds zero and $\text{bos}+8$ holds the unit number. The executing order is that (if any) which is currently being obeyed.

The bits of the order byte have the following meanings:

Bit 0	0=unit halted/inactive 1=actively obeying an order
Bit 1	0=move to target, but stop in adjacent square if target square occupied 1=attack (mêlée) or reinforce target
Bit 2	1=engage enemy encountered on route
Bit 3	1=report (see bits 4 to 7)
Bit 4	1=to report arrival at target
Bit 5	1=to report contacts
Bit 6	1=to report result of combat
Bit 7	1=acknowledge receipt of order

Reports are stored in stacks of 111 bytes, one stack for each unit (*brs* is the base address of the stack):

brs	Number of reports on the stack (1–5)	
$\text{brs}+1$	Delay on each report (turns)	
to		
$\text{brs}+5$		
$\text{brs}+6$	Turn report was made	} 1st report
$\text{brs}+7$	Report byte	
$\text{brs}+8$	Details of reporting unit at time	
to	report was made	
$\text{brs}+16$		
$\text{brs}+17$	Number of other friendly or enemy unit (if any)	} 2nd to 5th reports
$\text{brs}+18$	Details of other unit	
to		
$\text{brs}+26$		
$\text{brs}+27$	to $\text{brs}+47$	
$\text{brs}+48$	to $\text{brs}+68$	
$\text{brs}+69$	to $\text{brs}+89$	
$\text{brs}+90$	to $\text{brs}+110$	

The 'other unit' is a unit which has to be reported as a result of 'Report contacts' or 'Report result of combat' order. Details of units are stored in nine bytes allocated in the same order as in the army details (see Chapter 5).

The bits of the report byte have the following meanings:

Bit 0	0='other unit' is friendly 1='other unit' is enemy
Bit 1	
to	Not allocated
Bit 3	

- Bit 4 1=arrived at target
- Bit 5 1=in contact with . . .
- Bit 6 1=in combat with . . .
- Bit 7 1=order acknowledged

Omdurman 12

In 1896 the British government ordered the Egyptian Army, under its Sirdar, General Sir Herbert Kitchener, to commence the re-conquest of the Sudan. Ever since the murder of the Governor, General Charles Gordon, at Khartoum in January 1885, the Sudan had been under the control of the Mahdist forces. Their leader the Mahdi, died in June 1885 but was succeeded by Abullahi, the Khalifa. Khartoum lay in ruins and the Khalifa set up his capital on the other side of the Nile, at Omdurman. The Sudan was in disorder, and the Khalifa's army, known as the Dervishes, was a constant source of trouble on the borders between the Sudan and Egypt. The Dervishes under the Khalifa were a well-organised army of fierce tribesmen (Figure 12.1), divided into cohorts each commanded by an emir



Fig 12.1 A Dervish warrior.

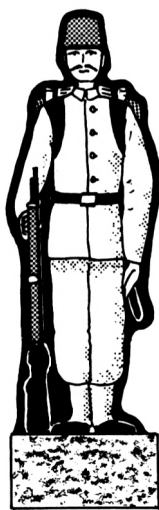


Fig 12.2 Egyptian infantry.

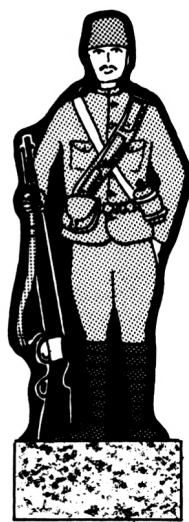


Fig 12.3 Egyptian cavalry and camel corps.

or other leader. It included a few cavalry, but mainly consisted of infantry. Most were armed with rifles and with the famous Dervish sword which they wielded with great dexterity. Not only did they exhibit considerable natural skill and cunning in battle but they were imbued with a fanaticism which made them an almost invincible enemy. At the Battle of Omdurman they outnumbered the Egyptian Army by over 2 to 1.

The Egyptian Army consisted of British-trained Egyptian and Sudanese regiments (Figure 12.2), commanded by British officers. It included cavalry and a corps of camelry (Figure 12.3). The army was accompanied by two British infantry brigades (Figure 12.4) and the 21st Lancers (Figure 12.5). There was further support from batteries of artillery, maxim guns and the cover of 10 gunboats on the Nile. The superior training, support and fire-power of the Sirdar's Anglo-Egyptian force helped to counterbalance the tremendous zeal and sheer numbers of the Khalifa's Dervishes.

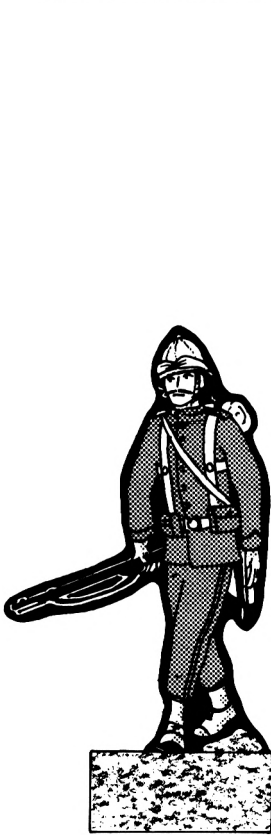


Fig 12.4 British infantry.

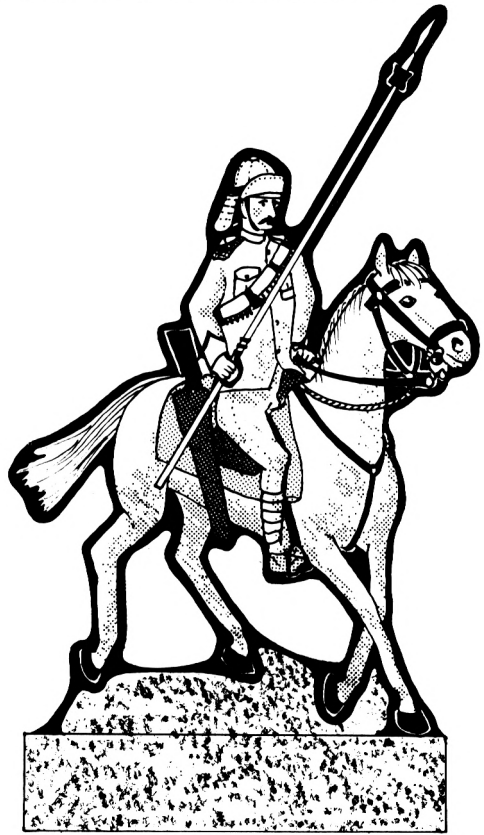


Fig 12.5 Trooper of the 21st Lancers.

The Sirdar's troops had had an arduous and eventful journey up the Nile from Cairo. They met with and overcame determined Dervish resistance on the way. Eventually they arrived in the vicinity of Omdurman on 1st September 1898. They set up their camp around the village of El Egeiga

on the banks of the Nile (Figure 12.6). Trenches were dug and thorn hedges erected to enclose the village and the surrounding area in a simple type of fortification known as a *zariba*. It was within this *zariba* that Kitchener's troops spent an uneasy night.

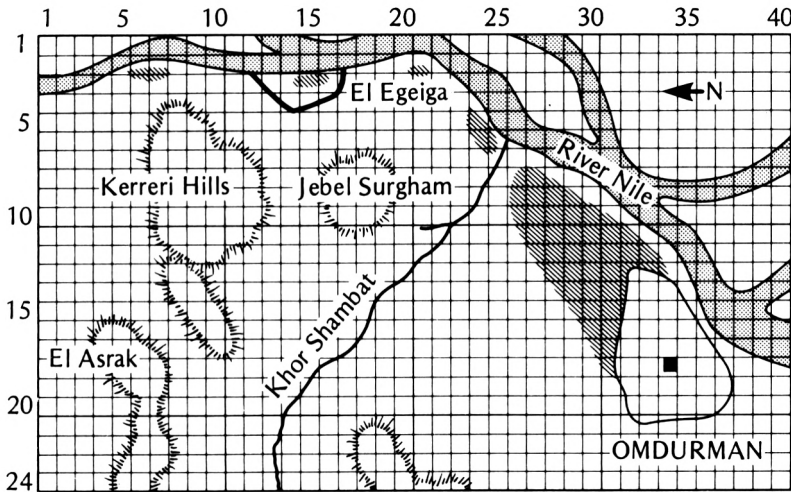


Fig. 12.6 The site of the battle of Omdurman. The black square in Omdurman is the Mahdi's tomb.

Hearing of the imminent arrival of Kitchener, and having suffered preliminary bombardment of Omdurman by gunboats, the Khalifa decided to move his troops out of Omdurman. They waited on the plain outside Omdurman. In the early hours of 2nd September they moved north, passing west of Jebel Surgham. The Khalifa mistakenly thought that a large body of Kitchener's forces were to the north behind the Kerrerri Hills. However, the Sirdar had sent only the Egyptian cavalry and the Camel Corps out from the *zariba* and the rest of his army was concentrated inside it.

While a large part of the Khalifa's army, under Ali Wad Helu and Sheikh al Din, the Khalifa's son, moved north behind the Kerrerri Hills and another part, under the Khalifa and his brother Yakub, remained behind Jebel Surgham, a large force led by Osman Azrak and Osman Digna swung round and charged the *zariba*. They were eventually beaten off, mainly by sustained artillery fire, and shelling from the gunboats. By about half past eight it seemed that the battle was over. The Egyptian cavalry and the Camel Corps were on their way back to the *zariba*. This is the stage at which the wargame begins.

Kitchener knew that the Khalifa had ordered most of his troops out of Omdurman. This had been a mistake on the Khalifa's part because, had they remained under the cover of the buildings of Omdurman, it would have been a formidable task for Kitchener's forces to dislodge them. Now was Kitchener's opportunity to occupy the city before the Dervishes could rally and return to Omdurman. Accordingly, Kitchener ordered his troops

to leave the zariba and march south to occupy Omdurman. An Egyptian brigade, commanded by 'Fighting Mac', Lieutenant-Colonel Hector Macdonald, was to act as rearguard. The 21st Lancers were sent ahead to reconnoitre and to prevent the Dervishes from returning to Omdurman. It was during this operation that the Lancers rode their famous charge against a horde of nearly 3000 Dervishes hidden in a dry watercourse.

The march on Omdurman was nearly a disaster for Kitchener. The enthusiasm of certain brigades to be the first to enter the city resulted in Macdonald's rearguard being left behind. He was attacked by large numbers of Dervishes returning from the Kerreri Hills and by others from behind Jebel Surgham. Had it not been for Macdonald's outstanding military ability and for reinforcements sent back from the advancing brigades, the final phase of the battle of Omdurman might well have been lost, and the Anglo-Egyptian forces annihilated.

The terrain

We visited the site of the battle early in the morning to avoid the searing heat of mid-day. The area is arid in the extreme, there being no vegetation except stunted thorn-trees or an occasional low-growing patch of Senna. The desert is stony with shifting patches of loose sand. Several times in a few kilometres we had to stop and pull our car out of the sandy drifts. It was still possible to find a spent bullet left from the battle. From the top of Jebel Surgham, we could see the Nile winding across the desert, until it disappeared over the horizon 20 miles away. Yet, although the air was clear at a height, close to the ground it was laden with desert dust. As our friend recounted the story of the battle of Omdurman to us, we imagined that we could really see the Dervish armies waiting in the haze for the battle to begin.

The game

The game begins with the troops in the positions they occupied at approximately 9 a.m. on 2nd September 1898. Army 1 is the Khalifa's army, consisting of six Dervish cohorts, represented by 17 units, each 3000 strong (Table 10). They are mainly infantry, carrying swords; about a quarter of the men are armed with rifles. The unit symbols are displayed in white on black (the same colours as the Khalifa's banner). The HQ symbol is a diagonal cross; the other unit symbols are marked with the unit number. Army 2 is Sirdar Kitchener's army. It consists of two British infantry brigades, represented by four units, each 2000 strong; five Egyptian brigades, represented by 10 units, each 1500 strong; the 21st Lancers (one unit, 400 strong); Egyptian cavalry (one unit, 400 strong); the Camel Corps (one unit, 400 strong). The unit symbols are displayed in red on blue. The HQ symbol is a diagonal cross and the other symbols bear unit numbers. The gunboats do not take part in the game as the main action at this stage of the battle was too far from the Nile. The game consists of 24

turns, each representing 15 minutes, a total of six hours. The map scale is 500 m to the square. Note that the map is oriented with north to the left. For this reason the customary 'NSEW' letters used when changing map section are replaced by arrow symbols.

Table 10 Khalifa's Army

Army name: Khalifa's

Paper no. 5: Pen no. 4: 1 type of unit

Type no.	Name and Commander	Type	Symb	Status	NULL no.	Str lo	Str hi	Row	Col
<i>Khalifa's HQ:</i>									
1	Khalifa	1	234	0	0	184	11	12	19
<i>Infantry cohorts:</i>									
2	Ali Wad Helu	1	235	4	0	184	11	14	5
3	Sheikh al Din	1	236	4	0	184	11	13	4
4	Sheikh al Din	1	237	4	0	184	11	13	5
5	Sheikh al Din	1	238	4	0	184	11	12	4
6	Sheikh al Din	1	239	4	0	184	11	12	5
7	Sheikh al Din	1	240	4	0	184	11	11	4
8	Sheikh al Din	1	241	4	0	184	11	11	5
9	Sheikh al Din	1	242	4	0	184	11	10	4
10	Yakub	1	243	0	0	184	11	16	15
11	Yakub	1	244	0	0	184	11	16	16
12	Yakub	1	245	0	0	184	11	15	17
13	Yakub	1	246	0	0	184	11	15	18
14	Khalifa	1	247	0	0	184	11	12	18
15	Osman Digna	1	248	0	0	184	11	11	21
16	Osman Azrak	1	249	0	0	184	11	14	18

Symb=symbol number; NULL=null detail; Str lo and Str hi=low and high bytes of strength.

Players take alternate turns, beginning with the Khalifa's army. Each turn consists of one phase during which the player issues orders, the units are moved and fight automatically, and reports are received. Chapter 11 describes what happens and how to play.

Each infantry unit has two movement points per turn, while each cavalry or camelry unit has three points. Open desert squares cost one point each, while jebel and hill squares, and the squares at the edge of Omdurman cost two points. Movement through the small villages beside the Nile takes two points, but the large inhabited area to the north of Omdurman counts as open desert. There is a khor, Khor Shambat, crossing the terrain, but there is no extra point cost for crossing this, as it is a relatively insignificant obstacle at this scale. There are no other rules, since most of the rules are necessarily incorporated into the independent action routine and have been dealt with in Chapter 11. As with other games, players are free to make up their own rules, though it may be less easy to put these into effect owing to the automatic nature of the routines.

At the end of each turn, there is a roster of the army which has just played. Note that this roster is based on *reported* information, and does not necessarily show the current status of each unit. Such is the Fog of War! You are given the opportunity to save the game at the end of each turn.

Victory The winner of the game is the first to occupy Omdurman. To occupy the city, the unit representing the leader (unit 1) must be within the boundary of Omdurman, and the total strength of all friendly units within Omdurman must be 50% more than the total strength of all enemy units within Omdurman. If turn 24 is reached without victory for one side or the other, the outcome of the game is inconclusive.

Setting up

It is assumed that you have already typed in the program of JUNGLE ATTACK (Chapter 6), and have played it several times. If so, you have a well-tested program that needs only a little alteration to turn it into the OMDURMAN program. If you have *not* already typed in JUNGLE ATTACK, type it in now, according to the instructions and listing given in Chapter 6. Note that you do not need the cipher program, JUNGLEC. When typing in JUNGLE ATTACK, you can save time by omitting the main program (lines 10–200) and certain subroutines that are not required for OMDURMAN. These are:

1000–1170	ADVANCE
1300–1460	DEPLOY
1470–1570	UNITLINE/ATTACK
1580–1650	COMMAND
1660–1760	CUSORO
1950–1990	INITIAL
2060–2260	PHASE
2320–2380	PROX
2390–2850	CLOSE
3260–3330	PATCH
3340–3410	MENDIT
3420–3920	FIRE
3930–3950	BLOCKED
3990–4270	MORTAR
4280–4370	PROJECTILE
4420–4600	LOS
4610–4630	DIGIN
4640–4760	SYMBOLS
4770–4790	HIDECOVER (if present)

When omitting these routines, simply omit to type them in, but *do not renumber* the program.

If you have already a working version of JUNGLE ATTACK, delete the main program (lines 10–200) and the subroutines listed above. *Do not renumber* the program.

Having obtained the bulk of the program in one of the ways described above, you need to make the following additions and amendments:

- 1 Type in the main program, OMDURMAN:

```

10 REM ** OMDURMAN **
20 PAPER 0:INK 0,1:PEN 1:INK 1,24:CLS
30 MEMORY 28823:CLS:GOSUB 3030:gtturn=2:cevf=2500:pe(3)
  =pa(1):pe(4)=pa(2):pa(3)=pe(1):pa(4)=pe(2):pa(5)=pa(
  3):pe(5)=pe(3):pa(6)=pa(4):pe(6)=pe(4):REM START
40 GOSUB 7290:REM STACKS
50 turn=PEEK(30000)+1:POKE 30000,turn
60 numarmy=turn-INT((turn-1)/PEEK(30001))*PEEK(30001)
70 na=numarmy:GOSUB 5370:REM MISSION
80 GOSUB 2890:REM TURNEND
90 IF na=1 THEN p2=2:ELSE p2=1
100 GOSUB 7660:REM HIDEIST
110 IF turn=49 THEN 140
120 IF mo=240 THEN 50
130 GOSUB 3000:REM SAVE
140 PAPER 0:PEN 1:MODE 1:LOCATE 2,2:PRINT"Game finish
  ed"
150 GOTO 150

```

- 2 Modify line 1230 of the DISPARMY subroutine:

```

1230 IF (PEEK(bu+4) AND 4) THEN 1280

```

- 3 Delete line 1240 from the DISPARMY subroutine.
 4 Type in this version of the UNITLINE subroutine:

```

1470 bu=aa(ny)+9*u+2:REM UNITLINE/OMDURMAN ***
1480 PEN #scr,pe(ny):PAPER #scr,pa(ny):LOCATE #scr,vx
  +2,vy:PRINT #scr,CHR$(PEEK(bu+3));
1500 SOUND 1,100,6
1510 PEN #scr,10:LOCATE #scr,vx+5,vy:PRINT #scr,MID$(
  STR$(FNdeek(bu+6)),2);
1570 RETURN

```

- 5 Modify line 2900 of the TURNEND subroutine:

```

2900 u1=1:WHILE u1<=units(na):CLS

```

- 6 Add lines 2922-2926

```

2922 bu=aa(na)+2+9*u:IF (PEEK(bu+4) AND 4) THEN ny=na
  +2:ELSE ny=na
2924 GOSUB 1470:REM UNITLINE
2926 bu=aa(na)+2+9*u:IF (PEEK(bu+4) AND 4) THEN bu=aa
  (na+2)+2+9*u

```

- 7 Amend line 3080 in the START subroutine:

```

3080 LOAD "OMDURD"

```

8 Type in this version of the SYMBOLS routine, noting that many of the lines are numbered in fives:

```

4640 CALL 29200,28824,234:REM SYMBOLS/ATTACK ***
4645 SYMBOL 234,0,60,90,102,102,90,60,0
4650 SYMBOL 235,0,28,4,8,16,28,0,0
4655 SYMBOL 236,0,28,4,28,4,28,0,0
4660 SYMBOL 237,0,20,20,28,4,4,0,0
4665 SYMBOL 238,0,28,16,28,4,28,0,0
4670 SYMBOL 239,0,28,16,28,20,28,0,0
4675 SYMBOL 240,0,28,20,4,4,4,0,0
4680 SYMBOL 241,0,28,20,28,20,28,0,0
4685 SYMBOL 242,0,28,20,28,4,28,0,0
4690 SYMBOL 243,0,46,42,42,42,46,0,0
4695 SYMBOL 244,0,36,36,36,36,36,0,0
4700 SYMBOL 245,0,46,34,36,40,46,0,0
4710 SYMBOL 246,0,46,34,46,34,46,0,0
4720 SYMBOL 247,0,42,42,46,34,34,0,0
4730 SYMBOL 248,0,46,40,46,34,46,0,0
4740 SYMBOL 249,0,46,40,46,42,46,0,0
4750 SYMBOL 250,0,46,42,34,34,34,0,0
4760 RETURN

```

9 Add the following new subroutines:

```

5370 top=1:lft=1:xc4=3:xn4=3:yc4=1:yn4=1:cr=3:REM MIS
SION ***
5380 IF na=1 THEN ne=2:p1=1:p2=2:ELSE ne=1:p1=2:p2=1
5390 ON BREAK GOSUB 7050:REM OPSTABLE
5400 CLS #1:PRINT #1,"TURN"INT((turn-1)/gturn+1);
5410 f$=aname$(na)+" army":GOSUB 3960:GOSUB 7050:REM
MESSAGE/OPSTABLE
5420 PAPER #3,5:PEN #3,4:CLS #3
5430 u=1:WHILE u>0
5440 u=-1:WHILE u<0 OR u>units(na):INPUT #3,"Unit";u$
:u=VAL(u$):WEND
5450 IF u=0 THEN 5660
5460 bos=ob(na)+24*(u-1):so=PEEK(bos):IF so<5 THEN PO
KE bos,so+1:ELSE f$="No couriers":GOSUB 3960:GOTO 566
0
5470 os=bos+6+3*so:POKE os,0:POKE os+1,0:POKE os+2,0
5480 ord=9:WHILE ord=9:obyte=0
5490 ord=2:WHILE ord>1 AND ord<8
5500 ord=0:WHILE ord<1 OR ord>9:INPUT #3,"Order";ord$
:ord=VAL(ord$):WEND
5510 IF ord=1 THEN obyte=0
5520 IF ord=2 THEN obyte=obyte OR 2
5530 IF ord=3 THEN obyte=obyte OR 4
5540 IF ord=4 THEN obyte=obyte OR 40
5550 IF ord=5 THEN obyte=obyte OR 72
5560 IF ord=6 THEN obyte=obyte OR 24
5570 IF ord=7 THEN obyte=obyte OR 136
5580 WEND:WEND
5590 IF ord=1 THEN 5650
5600 tc=-1:WHILE tc<0 OR tc>ncols:INPUT #3,"Column";t
c$:tc=VAL(tc$):WEND
5610 IF tc>0 THEN tr=0:WHILE tr<1 OR tr>nrows:INPUT #
3,"Row";tr$:tr=VAL(tr$):WEND:GOTO 5640
5620 army=0:WHILE army<1 OR army>2:INPUT #3,"Army";ar
my$:army=VAL(army$):WEND
5630 unit=0:WHILE unit<1 OR unit>units(army):INPUT #3
,"Tgt.unit";unit$:unit=VAL(unit$):WEND
5640 POKE os,obyte+1:IF tc=0 THEN IF army=na THEN POK
E os+1,unit:ELSE POKE os+2,unit:ELSE POKE os+1,tr:POK
E os+2,tc

```

```

5650 POKE bos+PEEK(bos),ROUND(SQR((PEEK(aa(na)+12)-PE
EK(aa(na)+3+9*u))^2+(PEEK(aa(na)+13)-PEEK(aa(na)+4+9*
u))^2)/(cr)+1
5660 WEND:CLS #3:f$="Accepted":GOSUB 3960
5670 FOR u=1 TO units(na):bos=ob(na)+24*(u-1):IF PEEK
(bos)=0 THEN 5690
5680 FOR k=1 TO PEEK(bos):POKE bos+k,MAX(PEEK(bos+k)-
1,0):NEXT
5690 NEXT
5700 IF RND(1)<0.05 THEN bos=ob(na)+24*(INT(RND(1)*un
its(na))):POKE (bos),MAX(PEEK(bos)-1,0)
5710 FOR u=1 TO units(na):bos=ob(na)+24*(u-1):bu=aa(n
a)+2+u*9
5720 IF (PEEK(bos)=0) OR ((PEEK(bu+4) AND 3)=3) THEN
5840
5730 fh=0:FOR k=1 TO PEEK(bos):IF PEEK(bos+k)=0 AND P
EEK(bos+3+k*3)=0 THEN fh=k
5740 NEXT:IF fh THEN POKE bos,PEEK(bos)-fh+1:FOR k=6
TO PEEK(bos)*3+5:POKE bos+k,PEEK(bos+k+(fh-1)*3):NEXT
:FOR k=1 TO PEEK(bos):POKE bos+k,PEEK(bos+k+fh-1):NEX
T
5750 k=1:WHILE PEEK(bos+k)>0 AND k<PEEK(bos)+1:k=k+1:
WEND
5760 IF k=PEEK(bos)+1 THEN 5840
5770 IF fh THEN POKE bos+21,0
5780 IF (PEEK(bos+21) AND 1) THEN 5840
5790 FOR kk=1 TO 3:POKE bos+20+kk,PEEK(bos+2+kk+k*3):
NEXT
5800 IF PEEK(bos)=k THEN 5840
5810 FOR byte=bos+3+k*3 TO bos+2+PEEK(bos)*3:POKE byt
e,PEEK(byte+3):NEXT
5820 FOR byte=bos+k TO bos+PEEK(bos)-1:POKE byte,PEEK
(byte+1):NEXT
5830 POKE bos,PEEK(bos)-1
5840 NEXT
5850 bu=aa(na+2)+11:bb=aa(na)+11:FOR u=1 TO units(na)
5860 IF (PEEK(bb+4) AND 4)=0 THEN FOR k=0 TO 8:POKE b
u+k,PEEK(bb+k):NEXT
5870 bu=bu+9:bb=bb+9:NEXT
5880 be=aa(ne+4)+11:bb=aa(ne)+11:FOR u=1 TO units(ne)
5890 IF (PEEK(bb+4) AND 4)=0 THEN FOR k=0 TO 8:POKE b
e+k,PEEK(bb+k):NEXT
5900 be=be+9:bb=bb+9:NEXT
5910 f$="Advancing":GOSUB 3960:REM MESSAGE
5920 FOR uu=1 TO units(na):bu=aa(na)+2+9*uu:bos=ob(na
)+24*(uu-1):brs=rb(na)+111*(uu-1):ur=PEEK(bu+1):uc=PE
EK(bu+2):tr=PEEK(bos+22):tc=PEEK(bos+23)
5930 IF (PEEK(bu+4) AND 4) THEN hf=1:ELSE hf=0
5940 IF (PEEK(bu+4) AND 3)=3 THEN 6760
5950 IF PEEK(bos+21)=40 THEN dir=INT(RND(1)*4)+1:GOTO
6460
5960 IF (PEEK(bos+21) AND 1)=0 THEN 6760
5970 IF tr>0 AND tc>0 THEN 6010
5980 IF tr THEN tr=PEEK(aa(na)+3+tr*9):tc=PEEK(aa(na)
+4+PEEK(bos+22)*9):bf=aa(na+2)+2+PEEK(bos+22)*9:IF ((
ABS(tr-ur)<5) AND (ABS(tc-uc)<5)) OR (uu=1) THEN 6010
:ELSE tr=PEEK(bf+1):tc=PEEK(bf+2):GOTO 6010
5990 tr=PEEK(aa(ne)+3+tr*9):tc=PEEK(aa(ne)+4+PEEK(bos
+23)*9):be=aa(ne+4)+2+PEEK(bos+23)*9:IF (ABS(tr-ur)<5
) AND (ABS(tc-uc)<5) THEN 6010:ELSE tr=PEEK(be+1):tc=
PEEK(be+2)
6000 IF tr=0 THEN tr=INT(RND(1)*nrows)+1:tc=INT(RND(1
)*ncols)+1
6010 IF PEEK(bu)>3 THEN all=3:ELSE all=2
6020 WHILE all>0:nm=0:ur=PEEK(bu+1):uc=PEEK(bu+2):IF
(PEEK(bu+4) AND 3)=3 THEN 6450

```

```

6030 byte=PEEK(bos+21) AND 136:IF byte=136 THEN GOSUB
7390:REM REPORT
6040 IF tc<>uc OR tr<>ur THEN 6120
6050 nz=ne:xz=wc:yz=wr:uf=0:uz=1:GOSUB 2270:REM FINDU
NIT
6060 IF uf=0 THEN 6090
6070 IF (PEEK(bos+21) AND 2) THEN cevt=1:GOSUB 7580:b
yte=PEEK(bos+21) AND 72:IF byte=72 THEN GOSUB 7390
6080 byte=PEEK(bos+21) AND 72:IF byte=72 THEN GOSUB 7
390
6090 byte=PEEK(bos+21) AND 24:IF byte=24 THEN GOSUB 7
390
6100 all=0:IF (PEEK(bos+21) AND 40=40) THEN POKE bos+
21,40:ELSE POKE bos+21,0
6110 GOTO 6450
6120 IF tc=uc THEN IF tr>ur THEN dir=3:ELSE dir=1
6130 IF tr=ur THEN IF tc>uc THEN dir=2:ELSE dir=4
6140 IF tc>uc AND tr>ur THEN IF RND(1)>0.5 THEN dir=2
:ELSE dir=3
6150 IF tc>uc AND tr<ur THEN IF RND(1)>0.5 THEN dir=2
:ELSE dir=1
6160 IF tc<uc AND tr>ur THEN IF RND(1)>0.5 THEN dir=4
:ELSE dir=3
6170 IF tc<uc AND tr<ur THEN IF RND(1)>0.5 THEN dir=4
:ELSE dir=1
6180 IF dir=1 THEN wr=ur-1:wc=uc
6190 IF dir=3 THEN wr=ur+1:wc=uc
6200 IF dir=2 THEN wr=ur:wc=uc+1
6210 IF dir=4 THEN wr=ur:wc=uc-1
6220 dirn=dir
6230 IF wr<1 OR wr>nrows OR wc<1 OR wc>ncols THEN 641
0
6240 nz=na:xz=wc:yz=wr:uf=0:uz=1:GOSUB 2270:REM FINDU
NIT
6250 bu=aa(na)+2+9*uu:bf=aa(na)+2+9*uz
6260 IF uf=0 THEN nz=ne:uz=1:GOSUB 2270
6270 bu=aa(na)+2+9*uu:be=aa(ne)+2+9*uz
6280 IF uf=0 THEN 6370
6290 byte=PEEK(bos+21) AND 40:IF byte=40 THEN GOSUB 7
390
6300 IF NOT(((PEEK(bos+21) AND 2)=2) AND (wc=tc) AND (
wr=tr)) THEN 6410
6310 IF nz=na THEN POKE bu+1,0:POKE bu+2,0:POKE bu+4,
3:str=FNdeek(bu+6)+FNdeek(bf+6):POKE bf+7,INT(str/256
):POKE bf+6,str-256*PEEK(bf+7):POKE bos,0:POKE bos+21
,0:POKE bu+6,0:POKE bu+7,0:IF (PEEK(bos+21) AND 40)=4
0 THEN byte=40:GOSUB 7390
6320 IF (nz=na) AND (hf=0) THEN bb=aa(na+2)+2+9*uu:PO
KE bb+1,0:POKE bb+2,0:POKE bb+4,3:POKE bb+6,0:POKE bb
+7,0
6330 IF nz=na THEN all=0:GOTO 6450
6340 cevt=0:GOSUB 7580:cp1=FNdeek(bu+6):cp2=FNdeek(be
+6)
6350 IF (PEEK(bos+21) AND 72)=72 THEN byte=72:GOSUB 7
390
6360 IF cp2>cp1 THEN all=0:GOTO 6450:ELSE GOTO 6410
6370 cost=PEEK(amap+wc-1+ncols*(wr-1)):IF all-cost<0
THEN 6410
6380 all=all-cost:POKE bu+1,wr:POKE bu+2,wc:IF uu=1
THEN POKE aa(na+2)+12,wr:POKE aa(na+2)+13,wc
6390 IF (hf=0) THEN bb=aa(na+2)+2+9*uu:POKE bb+1,wr:P
OKE bb+2,wc
6400 GOTO 6450
6410 IF nm=1 THEN all=0:GOTO 6450
6420 IF dirn=2 OR dirn=4 THEN IF RND(1)>0.5 THEN dir=
1:ELSE dir=3
6430 IF dirn=1 OR dirn=3 THEN IF RND(1)>0.5 THEN dir=
2:ELSE dir=4

```

```

6440 nm=1:GOTO 6180
6450 WEND
6460 IF (PEEK(bos+21) AND 40)<>40 THEN 6650
6470 byte=40:IF dir=2 OR dir=4 THEN 6570
6480 nz=na:xz=wc+1:yz=wr:uf=0:uz=1:GOSUB 2270
6490 IF uf THEN u=uz:GOSUB 7390
6500 nz=ne:uf=0:uz=1:GOSUB 2270
6510 IF uf THEN u=uz:GOSUB 7390
6520 nz=na:xz=wc-1:uf=0:uz=1:GOSUB 2270
6530 IF uf THEN u=uz:GOSUB 7390
6540 nz=ne:uf=0:uz=1:GOSUB 2270
6550 IF uf THEN u=uz:GOSUB 7390
6560 GOTO 6650
6570 nz=na:xz=wc:yz=wr+1:uf=0:uz=1:GOSUB 2270
6580 IF uf THEN u=uz:GOSUB 7390
6590 nz=ne:uf=0:uz=1:GOSUB 2270
6600 IF uf THEN u=uz:GOSUB 7390
6610 nz=na:yz=wr-1:uf=0:uz=1:GOSUB 2270
6620 IF uf THEN u=uz:GOSUB 7390
6630 nz=ne:uf=0:uz=1:GOSUB 2270
6640 IF uf THEN u=uz:GOSUB 7390
6650 IF (PEEK(bos+21) AND 4)=0 THEN 6760
6660 byte=PEEK(bos+21) AND 72:nz=ne:cevt=0:IF dir=2 O
R dir=4 THEN 6720
6670 xz=wc+1:yz=wr:uf=0:uz=1:GOSUB 2270
6680 IF uf THEN GOSUB 7580:IF byte=72 THEN GOSUB 7390
6690 xz=wc-1:uf=0:uz=1:GOSUB 2270
6700 IF uf THEN GOSUB 7580:IF byte=72 THEN GOSUB 7390

6710 GOTO 6760
6720 xz=wc:yz=wr+1:uf=0:uz=1:GOSUB 2270
6730 IF uf THEN GOSUB 7580:IF byte=72 THEN GOSUB 7390

6740 yz=wr-1:uf=0:uz=1:GOSUB 2270
6750 IF uf THEN GOSUB 7580:IF byte=72 THEN GOSUB 7390

6760 NEXT
6770 f$="All advanced":GOSUB 3960:REM MESSAGE
6780 FOR uu=1 TO units(na):brs=rb(na)+111*(uu-1):bu=a
a(na+2)+2+9*uu:IF PEEK(brs)=0 THEN 7030
6790 FOR k=1 TO PEEK(brs):POKE brs+k,MAX(PEEK(brs+k)-
1,0):IF PEEK(brs+k) THEN 6930
6800 ef=(PEEK(brs-14+k*21) AND 1)
6810 IF ef THEN be=aa(ne+4)+2+9*(PEEK(brs-4+21*k)):EL
SE bf=aa(na+2)+2+9*(PEEK(brs-4+21*k))
6820 FOR j=0 TO 8:POKE j+bu,PEEK(brs-13+21*k+j):NEXT

6830 IF ef THEN FOR j=0 TO 8:POKE j+be,PEEK(brs-3+21*
k+j):NEXT:ELSE FOR j=0 TO 8:POKE j+bf,PEEK(brs-3+21*k
+j):NEXT
6840 CLS#2:PRINT#2,MID$(STR$(uu),2)" at "MID$(STR$(PE
EK(bu+2)),2)"/"MID$(STR$(PEEK(bu+1)),2)" T"MID$(STR$(
PEEK(brs-15+k*21)),2);
6850 ny=na+2:u=uu:scr=3:vx=1:vy=1:GOSUB 1470:mo=0:WHI
LE mo<>32:GOSUB 4380:WEND:REM UNITLINE/MOVE
6860 CLS #3:byte=PEEK(brs-14+k*21)
6870 IF (byte AND 128) THEN PRINT #2,"Acknowledged";
6880 IF (byte AND 16) THEN PRINT #2,"Arrived";
6890 IF (byte AND 64) AND ef THEN PRINT #2,"In combat
";ny=ne+4:u=PEEK(brs-4+k*21)
6900 IF (byte AND 32) THEN PRINT #2,"In contact";u=P
EEK(brs-4+k*21):IF ef THEN ny=ne+4:ELSE ny=na+2
6910 GOSUB 1470:mo=0:WHILE mo<>32:GOSUB 4380:WEND:REM
UNITLINE/MOVE
6920 CLS #2:PAPER #3,5:CLS #3:f$="End of message":GOS
UB 3960:t=TIME:WHILE TIME<t+500:WEND
6930 NEXT
6940 k=0

```

```

6950 k=k+1:IF k>PEEK(brs) THEN 7030
6960 IF PEEK(brs+k)>0 THEN 6950
6970 IF PEEK(brs)=0 THEN 7030
6980 IF PEEK(brs)=k THEN 7010
6990 FOR byte=brs-15+k*21 TO brs-16+PEEK(brs)*21:POKE
    byte, PEEK(byte+21):NEXT
7000 FOR byte=brs+k TO brs+PEEK(brs)-1:POKE byte,PEEK
    (byte+1):NEXT
7010 POKE brs,PEEK(brs)-1
7020 GOTO 6960
7030 NEXT
7040 ON BREAK STOP:RETURN
7050 CLS #0:CALL 29003,30997+(top-1)*ncols*6+1ft*3,ex
    s:REM OPSTABLE ***
7060 IF na=1 THEN ne=2:ELSE ne=1
7070 GOSUB 1200:REM DISPARMY
7080 bb=aa(na)+11:bu=aa(na+2)+11:PEN pa(na):PAPER pe(
    na):FOR k=1 TO units(na)
7090 IF (PEEK(bb+4) AND 3)=3 THEN 7140
7100 IF (PEEK(bb+4) AND 4)=0 THEN 7140
7110 xs=PEEK(bu+2)-1ft+1:IF xs<1 OR xs>20 THEN 7140
7120 ys=(PEEK(bu+1)-top)*2+1:IF ys<1 OR ys>21 THEN 71
    40
7130 LOCATE xs,ys:PRINT CHR$(PEEK(bb+3));
7140 bb=bb+9:bu=bu+9:NEXT
7150 bb=aa(ne)+11:bu=aa(ne+4)+11:PEN pa(ne):PAPER pe(
    ne):FOR k=1 TO units(ne)
7160 IF (PEEK(bb+4) AND 3)=3 THEN 7210
7170 IF (PEEK(bb+4) AND 4)=0 THEN 7210
7180 xs=PEEK(bu+2)-1ft+1:IF xs<1 OR xs>20 THEN 7210
7190 ys=(PEEK(bu+1)-top)*2+1:IF ys<1 OR ys>21 THEN 72
    10
7200 LOCATE xs,ys:PRINT CHR$(PEEK(bb+3))
7210 bb=bb+9:bu=bu+9:NEXT
7220 CLS #4:PRINT #4,"NS EW":LOCATE #4,1,3:PRINT #4,"
    F"
7230 PEN 4:PAPER 7
7240 mo=0:WHILE mo=0 OR (mo>239 AND mo<244):GOSUB 43
    80:IF mo<>0 THEN GOSUB 1180
7250 GOSUB 1770:REM CURSOR4
7260 WEND
7270 IF xc4=3 AND yc4=3 THEN CLS #4:RETURN
7280 GOSUB 2000:GOTO 7070
7290 ob(1)=FNdeek(29997)+1:ob(2)=ob(1)+units(1)*24:RE
    M STACKS ***
7300 rb(1)=ob(2)+units(2)*24:rb(2)=rb(1)+units(1)*111

7310 aa(3)=aa(2)+11+units(2)*9:aa(4)=aa(3)+11+units(1
    )*9
7320 aa(5)=aa(4)+11+units(2)*9:aa(6)=aa(5)+11+units(1
    )*9
7330 final=rb(2)+units(2)*111-1:POKE 29998,INT(final/
    256):POKE 29997,final-256*PEEK(29998)
7340 IF PEEK(30000) THEN 7380
7350 POKE ob(1),0:CALL 29100,ob(1),ob(1)+1,final-ob(1
    ),0,1,29164
7360 CALL 29100,aa(1),aa(3),11+9*units(1),1,1,29164:C
    ALL 29100,aa(1)+14,aa(5)+14,units(1),9,9,29164
7370 CALL 29100,aa(2),aa(4),11+9*units(2),1,1,29164:C
    ALL 29100,aa(2)+14,aa(6)+14,units(2),9,9,29164
7380 RETURN
7390 brs=rb(na)+111*(uu-1):REM REPORT ***
7400 IF PEEK(brs)=5 OR RND(1)>.95 THEN RETURN
7410 POKE brs,PEEK(brs)+1:rs=brs+6+(PEEK(brs)-1)*21
7420 bu=aa(na)+2+9*uu:IF nz=na THEN bf=aa(na)+2+9*uz:
    ELSE bf=aa(ne)+2+9*uz
7430 POKE brs+PEEK(brs),ROUND(SQR((PEEK(aa(na)+12)-PE
    EK(bu+1))^2+(PEEK(aa(na)+13)-PEEK(bu+2))^2)/cr)+1

```

```

7440 POKE rs,INT((turn-1)/gturn+1)
7450 POKE rs+1,byte AND 240:IF ((byte=40) OR (byte=72
)) AND (nz=ne) THEN POKE rs+1,(PEEK(rs+1) OR 1)
7460 FOR j=0 TO 8:POKE rs+2+j,PEEK(bu+j):NEXT
7470 IF byte=136 THEN POKE bos+21,(PEEK(bos+21) AND 1
27):RETURN
7480 IF byte=24 THEN POKE bos+21,(PEEK(bos+21) AND 23
1):RETURN
7490 POKE rs+11,uz:IF nz=na THEN FOR j=0 TO 8:POKE rs
+12+j,PEEK(bf+j):NEXT:ELSE FOR j=0 TO 8:POKE rs+12+j,
PEEK(be+j):NEXT
7500 IF byte=40 THEN RETURN
7510 FOR j=0 TO 8:POKE rs+12+j,PEEK(bf+j):NEXT
7520 brs=rb(ne)+111*(uz-1)
7530 IF PEEK(brs)=5 OR RND(1)>0.95 THEN RETURN
7540 POKE brs,PEEK(brs)+1:rs=brs+6+(PEEK(brs)-1)*21
7550 POKE brs+PEEK(brs),ROUND(SQR((PEEK(aa(ne)+12)-PE
EK(be+1))^2+(PEEK(aa(ne)+13)-PEEK(be+2))^2)/cr)+1
7560 POKE rs,INT((turn-1)/gturn+1)
7570 POKE rs+1,65:FOR j=0 TO 8:POKE rs+2+j,PEEK(bf+j)
:NEXT:POKE rs+11,uu:FOR j=0 TO 8:POKE rs+12+j,PEEK(bu
+j):NEXT:RETURN
7580 cu1=uu:cu2=uz:bu=aa(na)+2+cu1*9:bb=aa(na+2)+2+cu
1*9:bc=aa(ne+4)+2+cu2*9:bd=aa(ne+2)+2+cu2*9:be=aa(ne
)+2+cu2*9:cp1=FNDEEK(bu+6):cp2=FNDEEK(be+6):GOSUB 2860
:REM RESCOM:REM COMBAT ***
7590 dcp1=MIN(ROUND((dcp1*(1+FNranorm/50))/cevf),cp1
):POKE bu+7,INT((cp1-dcp1)/256):POKE bu+6,cp1-dcp1-25
6*PEEK(bu+7):IF hf=0 THEN POKE bb+7,PEEK(bu+7):POKE b
b+6,PEEK(bu+6)
7600 dcp2=MIN(ROUND((dcp2*(1+FNranorm/50))/cevf),cp2
):POKE be+7,INT((cp2-dcp2)/256):POKE be+6,cp2-dcp2-25
6*PEEK(be+7):IF ABS(PEEK(aa(ne)+13)-PEEK(be+2))<6 AND
ABS(PEEK(aa(ne)+12)-PEEK(be+1))<6 THEN POKE bd+6,PEE
K(be+6):POKE bd+7,PEEK(be+7)
7610 IF hf=0 THEN POKE bc+6,PEEK(be+6):POKE bc+7,PEEK
(be+7)
7620 IF dcp1/cp1>0.3 THEN POKE ob(na)+24+(uu-1)*9,0:P
OKE bu+1,0:POKE bu+2,0:POKE bu+4,3:IF hf=0 THEN POKE
bb+1,0:POKE bb+2,0:POKE bb+4,3
7630 IF dcp2/cp2>0.3 THEN POKE ob(ne)+24+(uz-1)*9,0:P
OKE be+1,0:POKE be+2,0:POKE be+4,3:IF ABS(PEEK(aa(ne)
+13)-PEEK(be+2))<6 AND ABS(PEEK(aa(ne)+12)-PEEK(be+1))<
6 THEN POKE bd+1,0:POKE bd+2,0:POKE bd+4,3
7640 IF hf=0 THEN POKE bc+1,0:POKE bc+2,0:POKE bc+4,3

7650 RETURN
7660 hx=PEEK(aa(p2)+13):hy=PEEK(aa(p2)+12):REM HIDE DI
ST ***
7670 FOR j=1 TO 2:bu=aa(j)+11
7680 FOR k=1 TO units(j):IF ABS(hx-PEEK(bu+2))<6 AND
ABS(hy-PEEK(bu+1))<6 THEN POKE bu+4,PEEK(bu+4) AND 25
1:ELSE POKE bu+4,PEEK(bu+4) OR 4
7690 bu=bu+9:NEXT:NEXT:RETURN

```

Save the program under the file-name "OMDUR". Then load and save "MCODE" (see Chapter 3). This should be saved on to the same disk as "OMDUR" or immediately after it on tape. Finally, you need the data file which holds all the information about terrain and armies.

Readers who have typed in the wargaming utility programs should use these to build up the data file, as explained in the next section (Using the utilities).

Keying-in method

Readers who wish to get the game up and running without further ado should proceed as follows.

First type in this *cipher program*:

```

10 REM *** OMDURC ***
20 MEMORY 28823:CLS
30 PRINT"Storing army data"
40 FOR j=29997 TO 30328
50 READ x:POKE j,x
60 NEXT
70 PRINT"Storing map data"
80 READ x:POKE 30998,x:READ x:POKE 30999,x
90 j=31000:WHILE j<36759
100 READ x$:x=VAL("&" + x$):POKE j,(x AND 240)/16:POKE
j+1,x AND 15
110 READ x$:x$="&" + x$:POKE j+2,VAL(x$)
120 j=j+3:WEND
130 PRINT"Storing cover table"
140 READ x:POKE 37000,x:READ x:POKE 37001,x
150 PRINT"Storing movement table"
160 READ x:POKE 37002,x:READ x:POKE 37003,x
170 j=37004:WHILE j<37964:READ n:READ x
180 jj=j+n-1:FOR k=j TO jj:POKE k,x:NEXT
190 j=jj+1:WEND
200 PRINT"Storing CEV table"
210 FOR j=37964 TO 38037
220 READ x:POKE j,x
230 NEXT
240 SAVE "OMDURD",B,29997,8041
250 PRINT "OMDURD saved"
260 PRINT "Backup (Y/N)?"
270 a$="":WHILE a$<>"N" AND a$<>"Y"
280 a$=UPPER$(INKEY$):WEND
290 IF a$="Y" THEN 240
300 END

1000 DATA 252,165,3,0,2,58,117,213,117,0,0,0,0,75,72,65,76
1010 DATA 73,70,65,32,16,5,4,1,12,19,234,0,0,184,11,0,1
1020 DATA 14,5,235,4,0,184,11,0,1,13,4,236,4,0,184,11,0
1030 DATA 1,13,5,237,4,0,184,11,0,1,12,4,238,4,0,184,11
1040 DATA 0,1,12,5,239,4,0,184,11,0,1,11,4,240,4,0,184
1050 DATA 11,0,1,11,5,241,4,0,184,11,0,1,10,4,242,4,0
1060 DATA 184,11,0,1,16,15,243,0,0,184,11,0,1,16,16,244,0
1070 DATA 0,184,11,0,1,15,17,245,0,0,184,11,0,1,15,18,246
1080 DATA 0,0,184,11,0,1,12,18,247,0,0,184,11,0,1,11,21
1090 DATA 248,0,0,184,11,0,1,14,18,249,0,0,184,11,0,83,73
1100 DATA 82,68,65,82,32,32,17,6,3,2,6,17,234,4,0,220,5
1110 DATA 0,2,8,14,235,4,0,220,5,0,2,8,15,236,0,0,220
1120 DATA 5,0,2,5,15,237,4,0,220,5,0,2,4,15,238,4,0
1130 DATA 220,5,0,2,7,16,239,4,0,220,5,0,2,6,16,240,4
1140 DATA 0,220,5,0,2,5,17,241,4,0,220,5,0,2,3,15,242
1150 DATA 4,0,220,5,0,2,3,16,243,4,0,220,5,0,3,5,18
1160 DATA 244,4,0,208,7,0,3,4,18,245,4,0,208,7,0,3,5
1170 DATA 19,246,4,0,208,7,0,3,4,19,247,4,0,208,7,0,4
1180 DATA 7,22,248,4,0,144,1,0,5,5,4,249,4,0,144,1,0
1190 DATA 6,5,13,250,4,0,144,1,0
1200 DATA 24,40
1210 DATA 11,20,11,20,11,20,11,20,11,20,10,D6,10,D7,11,20
1220 DATA 11,20,11,20,10,D5,10,8F,10,8F,10,D7,11,20,10,D6,10,8F
1230 DATA 10,8F,10,8F,10,8F,10,8F,10,D7,11,20,11,20,10,D5,10,8F
1240 DATA 10,8F,10,D7,10,20,11,20,11,20,11,20,11,20,11,20,11,20
1250 DATA 11,20,11,20,11,20,11,20,11,20,12,20,12,20,11,20,11,20
1260 DATA 10,D6,10,8F,10,8F,10,D7,11,20,11,20,11,20,10,D5,10,8F
1270 DATA 10,8F,10,8F,10,8F,10,8F,10,8F,10,8F,10,8F,10,8F,10,8F

```

```

1280 DATA 10,D7,11,20,11,20,10,D5,10,8F,10,8F,10,D7,11,20,11,20
1290 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,01,8F
1300 DATA 12,20,11,20,11,20,10,D6,10,8F,10,D4,10,D5,10,8F,10,8F
1310 DATA 10,8F,10,8F,10,8F,10,8F,10,8F,10,8F,10,8F,10,D4
1320 DATA 11,20,11,20,11,20,10,8F,10,8F,11,20,11,20,10,D5
1330 DATA 10,8F,10,8F,10,D7,11,20,11,20,11,20,11,20,11,20
1340 DATA 11,20,11,20,11,20,11,20,11,20,11,20,D6,10,8F,10,D4
1350 DATA 15,88,15,88,10,D5,10,8F,10,8F,10,8F,10,8F,10,8F,10,8F
1360 DATA 10,8F,10,8F,10,D4,11,20,11,20,15,88,11,20,10,D5,10,8F
1370 DATA 10,D7,10,20,11,20,11,20,10,D5,10,8F,10,8F,11,20,11,20
1380 DATA 11,20,11,20,11,20,11,20,11,20,11,20,10,8F,10,8F
1390 DATA 10,8F,10,8F,10,D4,11,20,15,88,11,20,11,20,11,20,11,20
1400 DATA 17,C1,17,C3,15,88,15,88,11,20,17,C1,17,C3,11,20,11,20
1410 DATA 11,20,11,20,11,20,10,8F,10,8F,11,20,11,20,11,20,11,20
1420 DATA 10,8F,10,8F,11,20,11,20,11,20,11,20,11,20,11,20,11,20
1430 DATA 11,20,11,20,10,8F,10,8F,10,8F,10,D4,11,20,15,88,11,20
1440 DATA 11,20,11,20,11,20,11,20,11,20,17,C1,17,C3,11,20,15,88
1450 DATA 11,20,17,C5,11,20,11,20,11,20,11,20,10,D5,10,8F
1460 DATA 10,D7,11,20,11,20,11,20,10,8F,10,8F,11,20,11,20,11,20
1470 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
1480 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
1490 DATA 11,20,17,C1,17,C3,11,20,17,C2,17,C0,11,20,11,20,11,20
1500 DATA 11,20,11,20,11,20,10,8F,10,8F,11,20,11,20,11,20,10,8F
1510 DATA 10,8F,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
1520 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,19,CF
1530 DATA 11,20,11,20,11,20,11,20,11,20,11,20,17,C1,17,C6,17,C0
1540 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,10,8F,10,8F
1550 DATA 10,D7,11,20,11,20,10,8F,10,8F,11,20,11,20,11,20,11,20
1560 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
1570 DATA 11,20,11,20,19,DE,19,CF,19,CF,19,DF,11,20,13,20,13,20
1580 DATA 11,20,13,20,13,20,13,20,13,20,13,20,13,20,11,20,11,20
1590 DATA 11,20,15,88,10,D5,10,8F,10,8F,11,20,11,20,10,8F,10,8F
1600 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
1610 DATA 11,20,11,20,11,20,11,20,11,20,19,CF,19,CF,19,CF
1620 DATA 19,CF,11,20,1A,45,1A,6C,11,20,1A,45,1A,67,1A,65,1A,69
1630 DATA 1A,67,1A,61,11,20,11,20,11,20,11,20,11,20,10,8F,10,8F
1640 DATA 10,D7,11,20,10,8F,10,8F,11,20,11,20,11,20,11,20,11,20
1650 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
1660 DATA 11,20,19,D9,19,CF,19,CF,19,CF,19,DF,11,20,11,20,11,20
1670 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
1680 DATA 11,20,15,88,10,D5,10,8F,10,8F,10,D7,10,8F,10,8F,11,20
1690 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,D6,11,20
1700 DATA 11,20,11,20,11,20,11,20,11,20,19,D9,19,CF,19,CF,19,CF
1710 DATA 19,CF,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
1720 DATA 11,20,11,20,11,20,11,20,11,20,11,20,10,C2,10,D5,10,8F
1730 DATA 10,8F,10,8F,10,8F,11,20,11,20,11,20,11,20,11,20,11,20
1740 DATA 10,D6,10,8F,10,8F,11,20,11,20,11,20,11,20,11,20,13,20
1750 DATA 19,D9,19,CF,19,CF,19,CF,19,CF,19,DF,11,20,11,20,11,20
1760 DATA 11,20,19,DE,19,CF,19,DF,11,20,11,20,11,20,11,20,11,20
1770 DATA 10,C2,10,C0,11,20,10,8F,10,8F,10,8F,10,8F,11,20,11,20
1780 DATA 11,20,11,20,11,20,10,D6,10,8F,10,8F,10,8F,11,20,11,20
1790 DATA 11,20,11,20,11,20,1A,4B,1A,65,1A,72,1A,72,1A,65,1A,72
1800 DATA 1A,69,11,20,11,20,11,20,13,20,19,CF,19,CF,19,CF,11,20
1810 DATA 11,20,11,20,11,20,13,20,10,95,11,20,11,20,10,D5,10,8F
1820 DATA 10,8F,10,8F,11,20,11,20,11,20,10,20,10,D6,10,8F,10,8F
1830 DATA 10,8F,10,8F,11,20,11,20,11,20,11,20,11,20,19,D9
1840 DATA 19,CF,19,CF,19,CF,19,CF,19,CF,11,20,11,20,13,20,19,DE
1850 DATA 19,CF,19,CF,19,CF,11,20,11,20,11,20,11,20,11,20,10,95
1860 DATA 11,20,11,20,15,88,10,D5,10,8F,10,8F,10,D7,11,20,11,20
1870 DATA 10,D6,10,8F,10,8F,10,8F,10,8F,10,D4,11,20,11,20,11,20
1880 DATA 11,20,11,20,11,20,1A,4B,1A,69,1A,6C,1A,6C,1A,73,19,CF
1890 DATA 11,20,11,20,1A,4A,1A,65,1A,62,1A,65,1A,6C,11,20,11,20
1900 DATA 11,20,11,20,11,20,10,95,11,20,11,20,11,20,11,20,D5
1910 DATA 10,8F,10,8F,10,8F,10,8F,10,8F,10,8F,10,D4,11,20
1920 DATA 11,20,11,20,11,20,11,20,11,20,11,20,19,DD,19,CF
1930 DATA 19,CF,19,CF,19,CF,19,CF,19,DF,11,20,13,20,19,CF,19,CF
1940 DATA 19,CF,19,CF,13,20,13,20,11,20,11,20,10,C2,10,99,11,20
1950 DATA 11,20,15,88,11,20,11,20,10,8F,10,8F,10,8F,10,8F,10,8F

```

```

1960 DATA 10,8F,10,D4,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
1970 DATA 11,20,11,20,11,20,19,CF,19,CF,19,CF,19,CF,19,CF,19,CF
1980 DATA 11,20,1A,53,1A,75,1A,72,1A,67,1A,68,1A,61,1A,6D,11,20
1990 DATA 11,20,10,95,11,20,11,20,11,20,11,20,11,20,11,20,10,D5
2000 DATA 10,8F,10,8F,10,8F,10,8F,10,D4,11,20,11,20,11,20,11,20
2010 DATA 11,20,11,20,11,20,11,20,11,20,11,20,19,DE,19,CF,19,CF
2020 DATA 19,CF,19,CF,19,CF,19,CF,11,20,11,20,11,20,19,CF,19,CF
2030 DATA 19,CF,11,20,11,20,11,20,11,20,10,95,11,20,11,20,11,20
2040 DATA 11,20,11,20,15,88,11,20,10,8F,10,8F,11,20,11,20,11,20
2050 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2060 DATA 11,20,19,CF,19,CF,19,CF,19,CF,19,CF,19,CF,19,DC,11,20
2070 DATA 11,20,11,20,19,DD,19,CF,19,CF,11,20,11,20,11,20,11,20
2080 DATA 10,95,11,20,11,20,11,20,11,20,11,20,11,20,11,20,10,D5
2090 DATA 10,8F,10,D7,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2100 DATA 11,20,11,20,11,20,11,20,11,20,19,CF,19,CF,19,CF,19,CF
2110 DATA 19,CF,19,CF,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2120 DATA 11,20,11,20,11,20,10,C2,10,99,11,20,11,20,11,20,15,88
2130 DATA 11,20,15,88,11,20,11,20,10,8F,10,8F,11,20,11,20,11,20
2140 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2150 DATA 19,DD,19,CF,19,CF,19,CF,19,CF,19,DC,11,20,11,20,11,20
2160 DATA 11,20,11,20,11,20,11,20,11,20,11,20,10,C2,10,C0,11,20
2170 DATA 11,20,11,20,11,20,14,20,11,20,11,20,11,20,11,20,10,D5
2180 DATA 10,8F,10,D7,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2190 DATA 11,20,11,20,11,20,11,20,11,20,19,CF,19,CF,19,CF,11,20
2200 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2210 DATA 10,96,10,C0,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2220 DATA 11,20,11,20,15,88,11,20,10,8F,10,8F,11,20,11,20,11,20
2230 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2240 DATA 19,CF,19,CF,19,DC,11,20,11,20,11,20,11,20,11,20,11,20
2250 DATA 11,20,11,20,11,20,11,20,10,95,11,20,11,20,11,20,11,20
2260 DATA 11,20,11,20,11,20,15,88,15,88,11,20,11,20,11,20,10,D5
2270 DATA 10,8F,10,D7,11,20,11,20,11,20,10,D6,11,20,11,20,11,20
2280 DATA 11,20,11,20,11,20,11,20,19,CF,19,CF,11,20,11,20,11,20
2290 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,10,95
2300 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2310 DATA 11,20,1B,D6,1B,8F,1B,8F,10,8F,10,8F,10,8F,10,8F,10,8F
2320 DATA 10,8F,11,20,11,20,11,20,11,20,11,20,11,20,11,20,19,CF
2330 DATA 19,CF,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2340 DATA 11,20,11,20,11,20,10,95,11,20,11,20,11,20,11,20,11,20
2350 DATA 11,20,11,20,11,20,15,88,11,20,1B,8F,1B,8F,1B,8F,0B,D7
2360 DATA 10,8F,10,8F,10,8F,10,8F,10,8F,11,20,11,20,11,20,11,20
2370 DATA 11,20,11,20,11,20,19,DD,19,CF,19,DF,11,20,11,20,11,20
2380 DATA 11,20,11,20,11,20,11,20,11,20,11,20,10,C2,10,C0,11,20
2390 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,15,88
2400 DATA 1B,8F,1B,8F,1B,8F,1B,8F,10,8F,10,8F,10,8F,10,8F,10,D4
2410 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,19,CF
2420 DATA 19,CF,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2430 DATA 10,C2,10,C0,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2440 DATA 11,20,11,20,11,20,11,20,11,20,1B,8F,1B,8F,1B,8F,10,8F
2450 DATA 10,8F,10,8F,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2460 DATA 11,20,11,20,11,20,19,CF,19,CF,19,DF,11,20,11,20,11,20
2470 DATA 11,20,11,20,11,20,10,C2,10,C0,11,20,11,20,11,20,11,20
2480 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,15,88,1B,8F
2490 DATA 1B,8F,1B,8F,1B,8F,0B,D7,10,8F,10,8F,11,20,11,20,11,20
2500 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,19,DD,19,CF
2510 DATA 19,CF,11,20,11,20,11,20,11,20,11,20,10,C2,10,C0,11,20
2520 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2530 DATA 11,20,15,88,11,20,1B,8F,1B,8F,1B,8F,1B,8F,1B,8F,10,8F
2540 DATA 10,8F,10,D7,11,20,11,20,11,20,11,20,19,DE,19,CF,19,DF
2550 DATA 11,20,11,20,11,20,19,CF,19,DF,11,20,11,20,11,20,11,20
2560 DATA 13,20,10,95,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2570 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,1B,8F,1B,8F
2580 DATA 1B,8F,1B,8F,1B,8F,10,8F,10,8F,10,8F,10,8F,11,20,11,20
2590 DATA 11,20,19,CF,19,CF,19,CF,11,20,11,20,11,20,19,DD,19,DF
2600 DATA 11,20,11,20,11,20,13,20,13,20,10,95,11,20,11,20,11,20
2610 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2620 DATA 11,20,11,20,1B,8F,1B,8F,1B,8F,1B,8F,1B,8F,0B,D7,10,8F
2630 DATA 10,8F,10,8F,11,20,11,20,11,20,19,CF,19,CF,19,CF,19,CF

```

```

2640 DATA 11,20,11,20,11,20,19,CF,11,20,11,20,13,20,13,20,11,20
2650 DATA 10,95,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2660 DATA 11,20,11,20,11,20,11,20,11,20,18,D6,1B,8F,1B,8F,1B,8F
2670 DATA 1B,8F,1B,8F,1B,8F,10,8F,10,8F,10,8F,11,20,11,20,11,20
2680 DATA 19,DD,19,CF,19,CF,19,CF,11,20,11,20,11,20,19,CF,11,20
2690 DATA 11,20,13,20,11,20,10,C2,10,C0,11,20,11,20,11,20,11,20
2700 DATA 11,20,11,20,11,20,11,20,11,20,14,20,11,20,11,20,11,20
2710 DATA 1B,8F,1B,8F,1B,8F,1B,8F,1B,8F,1B,8F,1B,8F,10,D5,10,8F
2720 DATA 10,8F,11,20,11,20,11,20,11,20,19,CF,19,CF,19,CF,11,20
2730 DATA 11,20,11,20,11,20,11,20,11,20,11,20,10,C2,10,C0,11,20
2740 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2750 DATA 11,20,11,20,11,20,11,20,1B,8F,1B,8F,1B,8F,1B,8F,1B,8F
2760 DATA 1B,8F,1B,8F,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2770 DATA 19,DD,19,CF,19,CF,11,20,11,20,11,20,11,20,11,20,11,20
2780 DATA 10,C2,10,C0,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2790 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,1B,D5
2800 DATA 1B,8F,1B,8F,1B,8F,82,0E,1B,8F,1B,8F,11,20,11,20,11,20
2810 DATA 11,20,11,20,11,20,11,20,11,20,19,CF,19,CF,11,20,11,20
2820 DATA 11,20,11,20,11,20,11,20,10,95,11,20,11,20,11,20,11,20
2830 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2840 DATA 11,20,11,20,11,20,11,20,1B,8F,1B,8F,1B,8F,1B,8F,1B,8F
2850 DATA 1B,8F,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2860 DATA 19,CF,19,DB,11,20,11,20,11,20,11,20,11,20,11,20,10,95
2870 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2880 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,1B,D5
2890 DATA 1B,8F,1B,8F,1B,8F,1B,8F,1B,8F,11,20,11,20,11,20,11,20
2900 DATA 11,20,11,20,11,20,11,20,19,CF,19,DB,11,20,11,20,11,20
2910 DATA 11,20,11,20,11,20,10,95,11,20,11,20,11,20,11,20,11,20
2920 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2930 DATA 11,20,11,20,11,20,11,20,1B,8F,1B,8F,1B,8F,1B,8F,1B,8F
2940 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,19,CF
2950 DATA 19,DB,11,20,11,20,11,20,11,20,11,20,11,20,10,95,11,20
2960 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
2970 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,1B,D5
2980 DATA 1B,8F,1B,8F,1B,8F,1B,D4,11,20,11,20,11,20,11,20,11,20
2990 DATA 11,20,11,20,11,20,19,CF,11,20,11,20,11,20,11,20,11,20
3000 DATA 11,20,10,C2,10,C0,11,20,11,20,11,20,11,20,11,20,11,20
3010 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3020 DATA 11,20,13,20,13,20,13,20,13,20,13,20,13,20,13,20,13,20
3030 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,19,CF,11,20
3040 DATA 11,20,11,20,11,20,11,20,11,20,10,95,11,20,11,20,11,20
3050 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3060 DATA 11,20,11,20,11,20,11,20,1B,4F,1B,6D,1B,64,1B,75
3070 DATA 1B,72,1B,6D,1B,61,1B,6E,11,20,11,20,11,20,11,20,11,20
3080 DATA 11,20,11,20,19,CF,19,CF,19,DB,11,20,11,20,11,20,11,20
3090 DATA 10,95,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3100 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3110 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3120 DATA 11,20,11,20,11,20,11,20,11,20,11,20,19,CF,19,CF,19,DB
3130 DATA 11,20,11,20,11,20,11,20,10,95,11,20,11,20,11,20,11,20
3140 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3150 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3160 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3170 DATA 19,DE,19,CF,19,CF,11,20,11,20,11,20,11,20,11,20,10,95
3180 DATA 11,20,11,20,11,20,11,20,11,20,19,DE,19,DF,11,20,11,20,11,20
3190 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3200 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3210 DATA 11,20,11,20,11,20,11,20,19,CF,19,CF,19,CF,11,20,11,20
3220 DATA 11,20,11,20,11,20,10,95,11,20,11,20,11,20,11,20,19,CF
3230 DATA 19,CF,19,DF,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3240 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3250 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,19,CF
3260 DATA 19,CF,19,CF,11,20,11,20,11,20,11,20,11,20,10,95,11,20
3270 DATA 11,20,11,20,11,20,19,DD,19,CF,19,CF,11,20,11,20,11,20
3280 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3290 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3300 DATA 11,20,11,20,11,20,19,CF,19,CF,19,DC,11,20,11,20,11,20
3310 DATA 11,20,11,20,10,95,11,20,11,20,11,20,11,20,11,20,19,CF

```

```

3320 DATA 19,CF,19,CF,19,CF,19,DF,11,20,11,20,11,20,11,20
3330 DATA 11,20,11,20,11,20,11,20,11,20,11,20,11,20,11,20
3340 DATA 11,20,11,20,11,20
3350 DATA 0,0
3360 DATA 24,40
3370 DATA 46,9,2,2,11,9,1,1,1,2,23,9,1,1,2,2,6,1,3,2,7,1,17,9
3380 DATA 7,1,1,2,16,1,16,9,6,1,4,2,13,1,1,2,16,9,6,1,5,2,13,1
3390 DATA 1,2,15,9,6,1,6,2,4,1,3,2,8,1,13,9,6,1,6,2,3,1,4,2
3400 DATA 10,1,11,9,6,1,7,2,2,1,4,2,11,1,10,9,6,1,7,2,3,1,3,2
3410 DATA 12,1,9,9,6,1,6,2,20,1,8,9,7,1,3,2,23,1,7,9,7,1,2,2
3420 DATA 22,1,3,2,6,9,7,1,3,2,21,1,1,2,2,1,1,2,5,9,8,1,3,2
3430 DATA 20,1,1,2,2,1,1,2,5,9,3,1,3,2,3,1,2,2,20,1,1,2,3,1
3440 DATA 1,2,4,9,3,1,4,2,3,1,1,2,19,1,1,2,5,1,1,2,3,9,4,1,3,
2,23,1
3450 DATA 1,2,5,1,1,2,8,1,2,2,24,1,1,2,4,1,1,2,8,1,2,2,25,1
3460 DATA 5,2,8,1,2,2,10,1,2,2,26,1,3,2,9,1,3,2,24,1,3,2,10,1
3470 DATA 3,2,24,1,3,2,11,1,5,2,17,1
3470 DATA 12,6,0,4,4,6,6,7,40,0,0,0,0,0,32,0,0
3480 DATA 0,0,0,32,0,0,0,0,0,28,0,0,0,0,0,25,0
3490 DATA 0,0,0,0,0,140,130,80,90,100,100,0,0,0,0,0,110
3500 DATA 0,0,0,0,0,140,0,0,0,0,0,130,0,0,0,0,0
3510 DATA 110,0,0,0,0,0,0

```

When you have typed in this program, check it extremely carefully and then save it under the name "OMDURC".

With disks Save OMDURC on the same disk as the other programs. We use it to create a *data file*, called "OMDURD". To create this file, all you have to do is to run OMDURC. OMDURC automatically creates the data file OMDURD on the same disk as itself and the other programs (OMDUR and MCODE). OMDURD is the data file which is loaded by the game program, OMDUR, when it is run.

With tapes The procedure is slightly different. Save OMDURC on a separate tape. Now load OMDURC. Then place the tape which has OMDUR and MCODE already on it, in the recorder, wound on to just beyond the end of the recording of MCODE. Now run OMDURC. Press 'RECORD' and 'PLAY' and any keyboard key when asked to do so. The data file, OMDURD, is then saved on the tape, immediately after MCODE. OMDURD is the data file which is loaded by the game program, OMDUR, when it is run.

After you have used OMDURC to create OMDURD, you may never need to use it again. However, it is best to keep the recording in case you have made mistakes in entering the data. It may then be necessary to edit OMDURC.

With OMDUR, MCODE and OMDURD on the tape (in that order), or on the same disk, you are ready to play. Type RUN "OMDUR" and press ENTER. OMDUR is loaded and run; it then loads MCODE and OMDURD, after which the game begins.

Using the utilities

For detailed instructions in using the utilities, see the earlier chapters concerned. Below we set out a suggested plan for putting together the data file for OMDUR.

1 RECRUITER: Prepare a file for each army, named "Khalifa's" and "Sirdar's" respectively, and save them under KHALIFA and SIRDAR. The data required is in Tables 10 and 11. Note that there is one type of unit in the Khalifa's army; all are of type 1. The Sirdar's army is made up of types 2 to 6 so there are six types (none of type 1). Starting positions are specified, and conform as closely as possible to the historical deployment of both sides at the commencement of the second phase of the battle (about 8.30 a.m.). If you prefer, you need not specify starting positions, in which event insert two deployment phases near the start of the game program, and type in the subroutines DEPLOY, INITIAL, MENDIT and NEWMAP.

Table 11 Sirdar's army

Army name: Sirdar's

Paper no. 6: Pen no. 3: 6 types of unit

Unit no.	Type name and Commander	Type	Symb no.	Status	NULL	Str lo	Str hi	Row	Col
<i>Sirdar's HQ:</i>									
1	Maxwell's Brigade	2	234	4	0	220	5	6	17
<i>Egyptian/Sudanese Division:</i>									
2	Macdonald's Brigade	2	235	4	0	220	5	8	14
3	Macdonald's Brigade	2	236	0	0	220	5	8	15
4	Laurie's Brigade	2	237	4	0	220	5	5	15
5	Laurie's Brigade	2	238	4	0	220	5	4	15
6	Lewis' Brigade	2	239	4	0	220	5	7	16
7	Lewis' Brigade	2	240	4	0	220	5	6	16
8	Maxwell's Brigade	2	241	4	0	220	5	5	17
9	Collinson's Brigade	2	242	4	0	220	5	3	15
10	Collinson's Brigade	2	243	4	0	220	5	3	16
<i>British Division (General Gatacre):</i>									
11	Wauchope's Brigade	3	244	4	0	208	7	5	18
12	Wauchope's Brigade	3	245	4	0	208	7	4	18
13	Lyttleton's Brigade	3	246	4	0	208	7	5	19
14	Lyttleton's Brigade	3	247	4	0	208	7	4	19
<i>Cavalry and camelry:</i>									
15	Martin's 21st Lancers	4	248	4	0	144	1	7	22
16	Broadwood's Egyptian Cavalry	5	249	4	0	144	1	5	4
17	Hopkinson's Camel Corps	6	250	4	0	144	1	5	13

Symb=symbol number; NULL=null detail; Str lo and Str hi=low and high bytes of strength.

There are four details. The values in the 'Status' column have the value '0' or '4' indicating which units are hidden to begin with. The second detail is a 'null' detail. The values in the two 'Strength' columns are the actual strengths of the units, as expressed by two bytes. Detail 3 is the low byte and detail 4 is the high byte.

2 CARTOGRAPHER. The terrain map has 40 columns and 24 rows (Figure 12.6). It is better if the text characters on the map are in the lower

half of the square. The colours in which we mapped the various types of terrain are: open desert = bright yellow; high ground (hills and jebels) = bright yellow and yellow (appears olive green) using chequered symbols such as ASCII 216 (see User Instructions), trenches and hedges of the zariba = bright magenta, River Nile and Khor Shambat = blue, villages and the settled area north of Omdurman = black, Omdurman = pink, text letters = pastel blue. Save the finished map under the name "OMDMAP".

3 **TABLER.** Two tables are needed. There is no cover table in this game. If we start the first table (OMDURM) at 37002, two bytes (31000 & 31001) are automatically left as zeros, signifying zero column and zero row, for the missing cover table.

OMDURM holds movement costs and begins at 37002; movement costs are normally one for each square. The hill and jebel squares, the *borders* of Omdurman and the villages, cost two points. The settled area to the north of Omdurman counts as desert, with a cost of one. The squares along the south edge of the Nile should be nine, so that units are unable to move on to or cross the Nile. The squares to the east of the '9' squares (above them on the map) are '0', since units never move on that area of the map.

OMDCEV, the CEV table, has 6 rows and 12 columns and begins at 37964, use the values given in Table 12.

Table 12 OMDCEV

Combat type	Attacking unit type	Attacked unit type					
		1	2	3	4	5	6
0:range=1	1	0	4	4	6	6	7
	2	40	0	0	0	0	0
	3	32	0	0	0	0	0
	4	32	0	0	0	0	0
	5	28	0	0	0	0	0
	6	25	0	0	0	0	0
1:Mêlée	1	0	140	130	80	90	100
	2	100	0	0	0	0	0
	3	110	0	0	0	0	0
	4	140	0	0	0	0	0
	5	130	0	0	0	0	0
	6	110	0	0	0	0	0

INTELLIGENCE. Use this program to load and store the following files:

Army 1 KHALIFA
 Army 2 SIRDAR
 Map OMDMAP
 Table 1 OMDURM
 Table 2 OMDCEV

The complete data file is saved under the name OMDURD. With disks, it should be on the same disk as the OMDUR and MCODE files. With tape, it should follow after OMDUR and MCODE.

Notes for programmers

The sequence of the main program is:

```
20  Set screen colours.
30  Reserve memory; call START to initialise the game.
40  Call STACKS to set up data stacks for MISSION.
50  Calculate turn number.
60  Determine number of army to play this turn (phasing army).
70  Call MISSION.
80  Call TURNEND; Save or continue?
90-100 Set p1/p2. Call HIDE DIST to hide units of phasing army, ready
for next turn.
110  To end game after move 24.
120  Back to line 50, if continuing.
130  Save game.
140-150 Restore mode 1 and its colours. End program.
```

Subroutines

The following subroutines appear for the first time in this program. Variables and arrays are defined in Appendix A. *MISSION*, *OPSTABLE*, *STACKS* and *REPORT*, are described in Chapter 11.

HIDE DIST Uses *PROX* to detect units within a given distance (*lim*) of the HQ unit. These are made visible by making the 'hide' bit of the status byte equal to zero. Otherwise, it is set to '1' to make the unit invisible.

13 Two-computer wargaming

As has been demonstrated in earlier chapters, computers can bring greater realism into wargaming by introducing hidden movement and by simulating the Fog of War. The effect is even further enhanced when a wargame is played on two linked computers.

In two-computer wargaming, each player has sole use of a computer, from which his or her forces are commanded. The sequence of play is, in general, the same as in one-computer wargames but there are some important differences. One obvious difference is that it is no longer necessary for the players to change seats at particular points in each turn. As a result of this, and provided the computers are correctly positioned, it is impossible for players to 'accidentally' obtain a glimpse of the screen while their opponent is playing.

In a two-computer game, the phasing player (the one who is moving or firing) goes through the usual sequence of phases, such as 'Advance', 'Advance/Fire' and 'Close Combat'. While the phasing player is thus engaged, the non-phasing player (the one who is *not* actively moving or firing) is in 'View' phase. View phase is a special phase occurring only in two-computer wargames. During this phase, the screen displays the terrain map with unit symbols in the normal way. The non-phasing player may move the cursor around the screen, and may use the Roster routine to discover the status of friendly units. The map section may be changed in the normal way. But the non-phasing player is not able to move or fire units. At intervals, the map is updated automatically to show the latest movements of enemy units. The advantage of the View phase is that it gives the non-phasing player the opportunity to study the disposition of the armies, to observe the action of the enemy (unless hidden!) and to plan what to do next turn. When the phasing player has finished, a complete roster of troops is displayed, in the usual way. At the same time, on the other computer, the non-phasing player's View phase is terminated and a roster of his or her troops is displayed. In the following turn, the phasing

player becomes the non-phasing player and is in View phase. The non-phasing player becomes the phasing player, moving and firing units.

Equipment

The method of two-computer wargaming that we have developed for this book is based on the Pace Commstar RS232 interface (see Appendix B for suppliers). Our system may need to be adapted if a different make of RS232 interface is used. The equipment you require consists of an Amstrad CPC464, CPC664 or CPC6128 computer and a Pace Commstar interface. You also need a friend with an Amstrad computer of one of the above models (it need not be the same model as yours), and another Commstar interface. Finally, you need a special lead to connect the interfaces to each other.

The lead consists of a three-core cable, about four metres long for convenience. Each end of the cable has a 25-pin D-type socket, suitable for connecting to the D-type plug on the Commstar interface. The sockets are joined as follows:

Socket 1		Socket 2
Pin 2	to	Pin 3
Pin 3	to	Pin 2
Pin 7	to	Pin 7

Figure 13.1 shows the connections. The lead may be obtained ready-made from the supplier named in Appendix B.

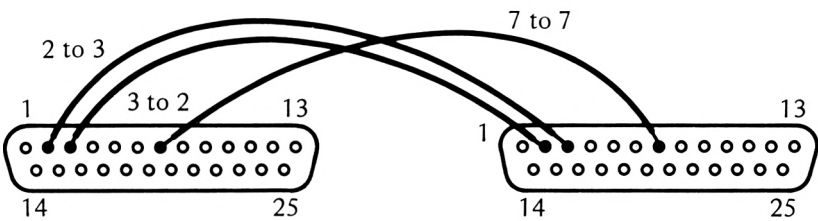


Fig 12.7 Three-wire connections between two RS232 interfaces (as seen from the rear side of the socket when connected to the plug on the interface).

To set up the system first position the computers so that the players are able to view only their own screen. Make sure that both computers are *switched off*. Attach an interface to the back of each computer. On the CPC464 it connects to the floppy disc port; on the CPC664 and CPC6128 it connects to the expansion port. Join the D-type plugs of the interfaces by the special cable. It does not matter which end of the cable connects to which interface. Finally switch on both computers. The system is now ready for loading the wargame programs.

Two-computer programs

It takes little time to adapt most wargame programs of the type described in this book to run on two computers. As an example, we shall describe how to adapt JUNGLE ATTACK. The method is similar for NASEBY. Programs using independent action (such as OMDURMAN) are long and require so much memory that it is not practicable to adapt them. The principle of the two-computer programs is that, after each phase played by the phasing player, the computer goes to a TALK subroutine. This copies the unit details from the memory of the phasing player's computer to the memory of the non-phasing player's computer. The non-phasing player is in View phase when these transfers are made. The operation of the phase is interrupted automatically for about 20 seconds while the transfer is occurring. After this, the non-phasing player may change map section to obtain updated information on the positions of units.

We require two programs, one for each army. We will call them the Army 1 program and the Army 2 program. In JUNGLE ATTACK, the Army 1 program is used by the Australian player. The Army 2 program is used by the Japanese player.

Army 1 program (Australian)

Begin with the program of JUNGLE ATTACK, prepared as in Chapter 6. It may have been adapted for hidden movement, as described in Chapter 7, but this makes no difference to the instructions below.

Modify the main program as follows:

- 1 Delete line 60, the line for Japanese deployment.
- 2 Type in this new line, which sends the Australian player into View phase when non-phasing:

```
105 IF (1 MOD numarmy) THEN GOTO 9000:REM VIEW
```

- 3 Add these lines to send the computer to the TALK routine to update the memory of the other computer:

```
115 GOSUB 9140:REM TALK
125 GOSUB 9140:REM TALK
145 GOSUB 9140:REM TALK
```

- 4 Modify line 1600:

```
1600 mo=0:WHILE (mo=0 OR (mo>239 AND mo<244)) AND f1<
3:GOSUB 4380:IF mo<>0 THEN GOSUB 1180
```

5 Type in these additional subroutines, noting that the vertical bar with which certain keywords begin is obtained by pressing SHIFT and the '@' key:

```

9000 f1=0:EVERY 100,3 GOSUB 9220:REM LISTEN:REM VIEW
***
9010 CLS #1:CLS #2:CLS #3:CLS #4:GOSUB 1950:REM INITI
AL
9020 PRINT#1,"TURN"INT((turn-1)/gturn+1)"View";
9030 IF numarmy=1 THEN na=2:ELSE na=1
9040 f$=aname$(na)+" army":GOSUB 3960:REM MESSAGE
9050 CLS #3:mo=0:WHILE mo<>32 AND mo<>114 AND f1<3:GO
SUB 1580:WEND:REM COMMAND
9060 IF f1=3 THEN 9130
9070 IF wind=4 THEN DI:GOSUB 2000:EI:GOTO 9050:REM NE
WMAF
9080 u=1:usq=0
9090 uf=usq:uz=u:nz=na:xz=xm:yz=ym:DI:GOSUB 2270:EI:u
=uz:xf=xm:yf=ym:usq=uf:REM FINDUNIT
9100 IF uf=0 THEN PRINT#3,"Roster complete";:t=TIME:W
HILE TIME<t+250:WEND:CLS #3:GOTO 9050
9110 ny=na:scr=3:vx=1:vy=1:DI:GOSUB 1470:EI:t=TIME:WH
ILE TIME<t+400:WEND:REM UNITLINE/ATTACK
9120 IF mo=114 THEN 9090
9130 IF f1=3 THEN EVERY 0,3 GOSUB 9220:f1=0:GOTO 150:
ELSE 9050
9140 LOCATE 2,2:PRINT"Updating memory":!COPEN:REM TAL
K ***
9150 bu=aa(1)+11:bf=aa(2)+11+9*units(2)
9160 tt=TIME:WHILE TIME<tt+600:!CPUT,0:WEND
9170 EVERY 4,2 GOSUB 9210:WHILE bu<bf
9180 WEND
9190 !CPUT,255
9200 !CCLOSE:EVERY 0,2 GOSUB 9210:RETURN
9210 DI:d%=PEEK(bu):!CPUT,d%:bu=bu+1:RETURN:REM SEND
***
9220 DI:d%=255:!CTIMEOUT,80:!COPEN:REM LISTEN ***
9230 bu=40000
9240 !CGET,@d%:IF d%=255 THEN !CCLOSE:RETURN
9250 !CGET,@d%:POKE bu,d%:bu=bu+1:IF d%<255 THEN 9250

9260 f1=f1+1:!CCLOSE
9270 b=40000:bytes=344
9280 IF PEEK(b)=0 THEN b=b+1:GOTO 9280
9290 CALL 29100,b,aa(1)+11,bytes,1,1,29164
9300 RETURN

```

Save the complete program. If you are using disks, you will need to save MCODE and ATTACKD on the same disk, as described in Chapter 6. If you are using tape save MCODE and ATTACKD on the same tape, immediately after the program.

Army 2 program (Japanese)

Begin with the program of JUNGLE ATTACK, prepared as in Chapter 6. It may have been adapted for hidden movement, as described in Chapter 7, but this makes no difference to the instructions below.

Modify the main program as follows:

- 1 Delete line 50, the line for Australian deployment.

2 Type in this new line, which sends the Japanese player into View phase when non-phasing:

```
105 IF (1 MOD numarmy)=0 THEN GOTO 9000:REM VIEW
```

3 Add lines 115, 125 and 145, as above to send the computer to the TALK routine.

4 Modify line 1600, as above.

5 Type in the additional subroutines, as above, but with a different version of line 9270:

```
9270 b=40000:IF turn=1 THEN bytes=90:ELSE bytes=344
```

Save the complete program and also MCODE and ATTACKD, as above.

The same procedures are used for modifying other games. If the game does not have deployment phases, all that has to be done to the main program is to type in the appropriate version of line 105 after the line which calculates *numarmy*. Then add the calls to the TALK subroutine at appropriate stages in the loop. Alter line 1600 and add the subroutines to complete the modification.

Sequence of play

We now describe how to play the two-computer version of JUNGLE ATTACK. Load and run the two programs. The screens display the map, with no units present. On Computer 1 (Australian Player) the message below the screen reads 'Deploy Aussie'. On Computer 2 the message is 'Deploy Japanese'. Players then deploy their own troops, in the usual way. When they have both finished, both move the cursor to 'F' and press the space bar. Computer 1 re-displays the map with the message 'Turn 1 — Advance Aussie'. Only the Australian army is visible on Computer 1 at this stage, since the memory has not yet been updated with information about the Japanese army. Computer 2 re-displays the map with the message 'Turn 1 — View Japanese'. Only Japanese units are visible.

The Australian player then moves units as in a normal advance phase. The Japanese player may view different sections of the map, and may roster any square, but cannot move units. Moving the cursor to 'F' and pressing the space bar does *not* end the View phase.

When the Australian player has finished advancing, the ADVANCE phase is terminated by moving the cursor to 'F' and pressing the space bar. The screen of Computer 1 clears and a message 'Updating memory' is displayed for about 20 seconds. While this is happening, the cursor on Computer 2 stops flashing and the display freezes.

The screen of Computer 1 then re-displays the map, with the message 'Turn 1 — A/Fire Aussie'. At the same time the cursor begins to flash again

on Computer 2. In turn 1 only there will still be no Japanese units visible on the screen of Computer 1, so firing will not be possible. While the Australian player is advancing, the Japanese player continues to view. If a map section is changed, the screen of Computer 2 will now reveal any Australian units that are visible to the Japanese side. The screen shows their positions as they were at the end of the Advance phase.

When the Australian player has finished advancing, the cursor is to be placed on 'F' again and the space bar pressed. Once again the 'Updating memory' message appears and the display on Computer 2 freezes.

If any units are in close combat, the information regarding this appears on the screen of Computer 1, while Computer 2 reverts to View mode. The Australian player reports the results to the Japanese player verbally. The Japanese player views the latest positions of the Australian troops. If any Japanese units have been killed in close combat, these units will now be 'missing' from the display!

When Close Combat is over, there is an automatic rally phase after which the memory of Computer 2 is updated for the third time. After this, both computers display a complete roster — Army 1 (Australian) on Computer 1 and Army 2 (Japanese) on Computer 2. Players press 'North' twice for the game to continue with turn 1, the Japanese player now becoming the phasing player. Computer 1 puts the Australian player into View phase while Computer 2 takes the Japanese player through the Advance, Advance/Fire and Close Combat phases. The Australian player sees Japanese units on the map for the first time after the Japanese player has completed the Advance phase of turn 1. This player-turn follows the same sequence as described above except that the roles of the two players and their computers are reversed. The game then continues to turn 2 and from there to the end at turn 20. In the description, we referred to memory being updated after the Advance/Fire phase and after the Rally Phase. If there are units in close combat, the Close Combat phase is put into action as described above. If there is no close combat, this phase is omitted and data is updated twice in succession, without a break. This explains why updating apparently takes twice as long when there is no close combat.

As in the one-player game, there is the option to save the game at the end of any player-turn. Each player may save the game or, if more convenient, either one of the players may save the game. In the latter case, the saved game is used by both players when resuming the game on a later occasion.

Notes for programmers

The subroutines used are:

VIEW Operates the View phase. It is a simplified form of PHASE and is not actually a subroutine, since it is reached by a GOTO and returns by way of a GOTO. Every two seconds (line 9000) it calls the LISTEN subroutine, to find out if information is being received at the interface. After three updating sessions (when $fl=3$, line 9130) it returns to line 150 of the main program.

TALK Controls the sending of data to the other computer. Data transmission begins with a stream of zeros lasting two seconds, to indicate to any listening computer that army data is to follow. *TALK* then calls *SEND* every 0.08 second to send one byte of army data. The complete data of both armies is sent, except for the first 11 bytes of army 1 (Army name and other data which is never changed). Finally a '255' is sent to indicate the end of transmission. The use of this protocol requires that the Army data does not begin with a zero. According to our format, it always begins with unit type number, which is always greater than zero. Also it must not contain '255' in any byte. Normally this presents no difficulty. If there is any possibility of a byte containing '255', the program can be altered (line 9190) to send some other value which it is known will not occur in the array data.

SEND Sends one byte of data and increments the data address.

LISTEN If no signal is detected within 0.08 second, it returns to the calling subroutine (*VIEW*). If it detects that data is arriving, it reads it and stores it in a buffer block in memory (from 40000 onward). Using a buffer overcomes the problem of synchronising the start of transfer of data between the two computers when using a three-wire system. When all data has been received (i.e. when the '255' has arrived), the routine stops listening. It then runs through memory from 40000 onwards until it finds the first non-zero byte (the first byte of army data), and calls the machine-code routine *BLOCK* to transfer the data from the buffer to the block of memory normally used for holding army details.

Modem wargaming 14

There are many enthusiastic wargamers who, for various reasons, are not able to meet with others to enjoy their hobby. Maybe there are no wargamers among their friends. Perhaps there is no wargaming society in the neighbourhood. Even if there is such a club, they may be unable to attend its meetings because of the times at which they are held, or owing to commitments which keep the would-be member at home. Others may prefer to concentrate on the tactics of the game in quiet and privacy.

For many such reasons, there is a large number of *solo wargamers*. The Solo Wargamers Association has been established to act as a means of contact (address, Appendix B). Communication between players is normally by post. One of the advantages of solo wargaming is that a player may allow ample time to consider each move. It is also possible to spend the intervals between receiving details of moves from one's opponent, in trying out several moves ahead to discover which tactic is the most likely to succeed. On the other hand, there is the disadvantage that postal contact involves recording unit positions and status exactly, without omissions, so that the recipient may know precisely what has happened at the sender's end. Recording moves for transmission by post can prove to be a tedious and exacting task. This is where the computer comes to the rescue!

With the development of inexpensive hardware for connecting computers to the telephone network, we have an ideal system for solo wargaming. The computer is programmed to inform the opponent's computer of the *exact* state of the game at the end of every turn. The moves made and the outcome of combat can be communicated to the opposing commander in a few seconds. Data is stored on tape or disk as it is received, so it is not necessary to proceed with the next turn immediately. The game may proceed as quickly or as slowly as the players wish.

Equipment

The method described in this chapter requires the equipment listed in Chapter 13, with the addition of a modem for each player. The modem we used in our trials is the Pace *Nightingale* modem (identical to the Amstrad

V21/23) though presumably modems of other types could be used instead. Both players also need to have a telephone network socket installed by British Telecom. The equipment is set up by plugging the Commstar RS232 interface on to the computer and connecting the modem to the interface by the cable provided. The mains lead of the modem is plugged into a mains outlet socket and the telephone lead is plugged into the BT telephone socket. Finally, the telephone is plugged into the socket on the rear panel of the modem.

Software

The Commstar interface has communications software, the Honeyterm software, already in its ROM. You also need to modify the wargames software to allow the state of the game to be saved to tape or disk at the end of every move. Here we describe how to modify JUNGLE ATTACK for solo wargaming by modem.

The programs for the two armies are slightly different, so we describe each in turn.

Army 1 program (Australian)

- 1 Add these lines 35 and 40 (new version).

```
35 IF PEEK(30000)=255 THEN POKE 30000,0:GOTO 70
40 IF a$="N" THEN 70
```

- 2 Delete line 60, so that only the Australian army deploys.
- 3 If you have previously installed hidden movement, as described in Chapter 7, delete lines 55, and 65, but leave line 145.
- 4 Add this line to save the game after deployment:

```
65 GOTO 180
```

If you have installed hidden movement, this line should be:

```
65 GOTO 145
```

- 5 Delete lines 160 and 170.
- 6 Alter line 200 to enter Honeyterm automatically:

```
200 t=TIME:WHILE TIME<t+2000:WEND:HT
```

7 Change line 1230 to:

```
1230 IF (PEEK(bu+4) AND 4) AND j=2 THEN 1280
```

The reason for doing this is explained later.

8 Amend line 3010:

```
3010 POKE 30032+maxu*18,PEEK(30000):SAVE file$,B,30010,23+maxu*18
```

9 Add line 3045:

```
3045 LOAD "ATTACKD"
```

10 Alter line 3070:

```
3070 IF a$="N" THEN PRINT:INPUT"File name";file$:LOAD  
UPPER$(file$)
```

11 Delete line 3080.

12 Alter line 3180:

```
3180 lwr=nrows-10:rgt=ncols-19:POKE 30000,PEEK(30032+  
maxu*18)
```

Army 2 program (Japanese)

1 Alter line 40 to:

```
40 IF a$="N" AND PEEK(30000) THEN 70
```

2 Delete line 50, so that only the Japanese army deploys.

3 As above.

4 Add this line to save the game after deployment:

```
65 POKE 30000,255:GOTO 180
```

If you have installed hidden movement add this line instead:

```
65 POKE 30000,255:GOTO 145
```

5 and 6 As above.

7 Change line 1230 to:

```
1230 IF (PEEK(bu+4) AND 4) AND j=1 THEN 1280
```

8 to 12 As above.

This version uses MCODE and the same ATTACKD data file as used by the normal version of the game. These should be present on the same tape or disk, as explained in Chapter 6.

Sequence of play

Unlike the two-computer wargaming system described in Chapter 13, a modem wargame does not require the two players to be simultaneously at their computers. All that is needed is that they should be in contact between turns, for the updating of the army data. There is no need to transfer the data immediately after saving. You can, for example, wait for the cheap rate. After data is transferred there is no need for the receiving player to play the next turn immediately. This is because the data is stored on tape or disk. The game can be re-run and the latest army data loaded whenever it is convenient for the player to play the next turn.

At the first session, the Australian player runs the program and deploys units in the normal way. As soon as the units have been deployed, the program goes to the routine described below which prepares a file of the army data, ready for transmission by telephone.

The screen clears and the message 'File name?' appears. Type in the file name, which must include the descriptor '.BIN'. Thus a suitable name to use after the Australian army has deployed might be 'AUSD.BIN'. The computer then saves a file of that name to tape or disk, depending on which Amstrad model is being used. The computer is then automatically put into Honeyterm mode. This makes ready to transmit the data. If it is not convenient to transmit at this time, simply press ESC, or switch off the computer. The file is safely stored on tape or disk and can be re-loaded for transmission on a later occasion.

Transmission of data

After deployment, the Australian transmits the new data to the Japanese player, as follows. Put the computer into Honeyterm mode (key 'HT' and press ENTER), if it is not already in that mode. Switch on the modem. Check that it is set to transmit and receive at 300 Baud and that it is in 8-bits

mode (the usual default conditions). The Australian player should have the tape or disk with the file on it ready in the recorder or drive. The Japanese player should also be in Honeyterm mode, with modem set to 300 Baud and 8-bits, and have a tape or disk ready in the recorder or drive for saving the received data.

At this point, one player telephones the other and, by voice, they make sure that they are both making preparations to transfer data. They agree what file name is to be used. The player who is originating the transmission should set the modem to 'Originate' while the other modem should be set to 'Answer'. The Australian player goes into transfer mode by pressing CTRL and '3'.

Select XMODEM, select SEND, then type in the file name (for example, 'AUSD.BIN'). The Japanese player also goes into transfer mode, selecting XMODEM and RECEIVE, and typing the same file name. Both players then set their modems to 'Modem connect'. When the tones of both modems are heard and the signal lamp which indicates 'carrier detect' is illuminated, they place their receivers on the telephone rests. Both players then press ENTER. The transmission of data is automatic, after a few seconds during which the modems are awaiting synchronisation. After a few more seconds the data will have been transferred to the receiving computer (Japanese). In JUNGLE ATTACK, there is only one block of data to be transferred, so the process is quickly completed. The Japanese player's computer then saves the data to tape or disk.

Converting the file The file is saved from an address in the Honeyterm buffer area, which is *not* the same as that at which the data is stored for use in a wargame. It is therefore necessary to convert the file so that it loads to the correct address. This is done by the receiving player as follows, with the computer in normal BASIC mode. Type in the following statements, pressing ENTER after each and take the appropriate action for loading and saving:

```
MEMORY 30000
LOAD "AUSD.BIN",30010
SAVE "AUSD.BIN",B,30010,383
```

Other file names may be used at different stages of the game or in different games, but they must always end in '.BIN'. The final number in the SAVE statement is the number of bytes of data transferred. This value is calculated from:

$23 + (\text{number of units in largest army}) \times 18$

For JUNGLE ATTACK the value is:

$23 + 20 \times 18 = 383$.

Next turn The newly created file may now be used for the next stage of the game. The Japanese player runs the game. The computer loads MCODE and the map and other data as usual.

Then the message 'New game?' appears. At this stage the data received from the Australian player has to be placed in memory, so that the Japanese player may see where Australian units are deployed. Place the tape or disk with the newly received data file on it in the recorder or drive. Key 'N'

(since this is no longer a *new* game) and press ENTER. The data will then be loaded. The screen then displays the map with the Australian units visible. The Japanese player deploys. After this, a file is saved as described above. Now it is the turn of the Japanese player to send data to the Australian player. The procedure is as described above, except that a different file name should be used, for example 'JAPD.BIN'.

Subsequent turns When the Australian player has run the program and used the data received from the Japanese player, units of *both* sides are displayed (except for hidden Japanese units if hidden movement is incorporated). The game is now at turn 1, with the Australian player having Advance phase, Advance/Fire phase and (possibly) Close Combat. After this, the game is saved to file, as described above and a new file (which might be called 'AUS1.BIN') is saved and then sent to the Japanese player. Note that, as well as the army data, the turn number is included in the data file, so that the game is always in the correct turn when re-run.

Play is described above as if it were a continuous sequence, but this is not necessarily so. Players may begin a new turn at any time after they have received the data of the previous turn, perhaps several days later. The essential point is that both sides must keep in step. The Australian is the first to deploy and the first to transmit data. The Japanese player deploys and transmits data next. After this they play/transmit alternately, starting with the Australian player.

View phase There is no specific View phase as there is in the two-player game of Chapter 13. Instead, the non-phasing player may re-run the program at any time while waiting for the phasing player to reply with fresh data. The non-phasing player uses the most recently transmitted file (after converting it to load at the correct position), as described earlier. It is possible not only to view the units but to continue the game into subsequent turns. This allows the non-phasing player to *explore* the possibilities of various manoeuvres. The program will, of course, end each turn of the game by going to the save routine. At this stage, do not enter a file name but press ESC twice. Then type 'GOTO 70' and press ENTER. The exploratory game then continues with the next player-turn. If this is the other (currently phasing) player's turn, you are able to move and fire units of the enemy army. You can test various tactics that you think the enemy might intend to employ. However, if hidden movement has been incorporated in the game, you will *not* be able to see hidden enemy units at any time. This is the effect of the alteration in line 1230. If you wish, you can save a file of the state of your exploratory game, but such a file should not be used as part of the normal game. When your opponent has finished moving, the data file sent to you is the one to be used when you resume the 'official' game. This will restore the game to its correct turn, with your units in the positions they held at the end of your previous 'official' turn, and your opponent's units in the positions to which they have just been moved.

Adapting other programs

Games such as NASEBY which do not begin with development phases are easier to adapt. Simply modify the corresponding lines of the program as

in items 5 to 11. With games such as OMDURMAN it is necessary to save additional army data and the entire order and report stacks.
Add this new line 45

```
45 POKE 30000,PEEK(aa(6)+units(2)*9+12)
```

Line 3010 becomes

```
3010 POKE aa(6)+units(2)*9+12,PEEK(30000):SAVE file$  
,B,30010,3*(22+9*(units(1)+units(2)))+1:file$=file$+"  
OR":SAVE file$,B,ob(1),(units(1)+units(2))*135
```

Line 3070 is altered to

```
3070 IF a$="N" THEN PRINT:INPUT"File name":file$:LOA  
D UPPER$(file$):file$=UPPER$(file$)+"OR":LOAD file$:G  
OTO 3090
```

There will be two files to transmit, the second having the same name as the first plus "OR" for "orders/records".

A Appendix

Variables and arrays

Variables

a\$ Input answer string.
add Address in memory.
all Movement points allowance.
amap Base address of movement allowance table.
anga Angle of elevation of projectile firing weapon.
angb Bearing of target from projectile firing unit (Clockwise, zero north).
army, army\$ Army to which target unit belongs.
ax, ay Absolute values of *dx* and *dy*.
b Address in data storage buffer.
b1, b2 Base address of army details, plus 1 or 2.
bb Base address of details of a unit.
bcev Base address of combat efficiency value table.
be Base address of details of an enemy unit.
bf Base address of details of a friendly unit.
bos Base address of order stack of a unit.
brs Base address of report stack of a unit.
bu Base address of details of a unit, usually friendly.
byte Address into which a value is to be POKEd, or value PEEKed from an address.
cdone 'Combat done' flag; -1 if a given instance of close combat has already been detected.
cevf The amount by which CEV values are multiplied when calculating the reduction in Combat Potential.
cevr Number of rows (and columns) in CEV table.
cevt Number of CEV table to be used in resolving combat.
cev1, cev2 Combat efficiency values of two opposing units.
char ASCII code of a character ('-' or arithmetic operator) typed in during BRIEFING.
cmap Base address of cover value table.

cn1, cn2 The number of units in each army in close combat.
co Cover value.
cost Movement points cost.
cp 'Charge phase' flag; -1 in charge phase.
cpf Factor by which combat potential is multiplied to obtain strength of a unit.
cp1, cp2 Combat potential of units of each army.
ctnr Number of unit randomly chosen for close combat when there are two or more units of any army on the same square.
ctn1, ctn2 The unit numbers of a pair of units in close combat.
d% Data byte received from or sent to another computer.
dcp1, dcp2 Decrease in combat potential of units of each army.
detail, detail\$ Number of a unit detail to be altered.
dfn Final unit to be deployed.
dir, dirn Direction of movement of independent unit.
done Flag set to 1 when an operation is complete.
dp 'Deploy' flag; -1 in a deploy phase.
dst First unit to be deployed.
dx, dy Difference in column number, or row number, between the map location of a firing unit and the map square it is firing at.
ea, eb Random errors in *anga* and *angb*.
ef 'Enemy' flag; -1 when 'other' unit is an enemy.
exs Amount by which number of columns in map exceeds 20.
f\$ Message for display by MESSAGE.
fb 'Blocked' flag; -1 if line of sight is blocked.
fh 'Halt' flag; -1 if halt order found.
file\$ File name.
final Last address used for storing data.
fl The number of times the computer has listened for data from another computer.
fp Fire phase flag; -1 if firing allowed.
gr Number of grenades carried.
gturn The number of player-turns in each game-turn.
hf 'Hidden' flag; -1 if unit hidden.
hx, hy Map location of first (HQ) unit of an army.
i Keeps count of the rows or columns between a firing unit and its target square, as the line of sight calculations proceed.
ix, iy Column or row intercepted by line of sight.
j, k, kk Loop indices.
j\$ Input command.
lft Number of left-most map column to be displayed.
lim The limiting number of rows or columns at or beyond which a unit is not proximate to a given unit.
loss1, loss2 Number of units lost by each army in close combat.
lwr Number of the topmost map row when the lowest map section is being displayed.
map Base address of map data (top left corner).
maxr Maximum range of a weapon, in squares.
maxu Number of units in the largest army.

mc Holds the number of a mortar-firing unit.
mf 'Move' flag; =0 if location from which a unit is being moved is not on the displayed map section.
mo Value returned by MOVE, the ASCII code of the key pressed.
mor The number of the mortar to be fired next.
mu Number of a mortar unit.
na The number of an army (usually friendly) being processed.
ncols Number of columns in the map.
ne The number of an enemy army being processed.
nm Number of moves tried unsuccessfully in independent movement.
nrows Number of rows in the map.
numarmy The number of the phasing army.
nxy The number of the column in array *xy* in which a pair of coordinates is to be stored.
ny Number of army, used in UNITLINE.
nz Number of army, used in FINDUNIT.
obyte Byte in which orders are coded.
ord Code number of an order.
os Base address of an order.
p Probability of a hit.
pb, pbu PEEKed byte, often ANDed to evaluate certain bits.
pdh, pdv Probable deviation from target centre in horizontal and vertical directions.
pe Value obtained by PEEKing.
pn Probability of a hit at half maximum range.
pw PEEKed value to determine if target is already wounded.
p1, p2 Army numbers used in combat routines.
r Random probability of a hit at range beyond half-range.
ra Target range, in squares.
rge Range of target of projectile.
rgt Number of left-most map column, when the right-most map section is being displayed.
rmax Maximum range of a projectile.
rs Randomly chosen value to determine the effect of a hit; base address of a report.
scr Number of window to be used for display.
sd Standard deviation of weapon (spread of hits).
sda, sdb Standard deviation of a projectile-firing weapon along and perpendicular to the line of fire.
so Number of orders on a stack.
sqb Base address of square details in map data.
t Terrain type. Value of TIME at start of delay loop.
tc, tc\$ Target column in independent action.
tf 'Too far' flag; =1 when unit moved too far.
top Number of topmost row of map to be displayed.
tpot Total combat potential lost by an army.
tr, tr\$ Target row of independent action.
tt The number of enemy units present on a target square, or value of TIME at start of a delay loop.

turn The current player-turn.

tx, ty Column and row of square being tested for blocking, in line of sight calculations.

u Number of unit.

uc Map column of location of independent unit.

ud Unit to be deployed.

uf 'Unit found' flag, set to 1 when FINDUNIT finds a unit on a given square.

unit, unit\$ Number of a target unit.

ur Map row of location of independent unit.

uu Number of unit.

uz Number of a unit found by FINDUNIT.

u1, u2 Two units being tested for proximity, or for mutual visibility.

value, value\$ Values typed in during BRIEFING.

vx, vy Screen coordinates at which message is to be located.

wc Map column of location to which independent unit is to move.

wind Window in which cursor is located.

wr Map row of location to which independent unit is to move.

xad, yad Map locations of squares adjacent to square of impact of a projectile.

xc, yc Location of cursor in window 0 (main screen).

xc4, yc4 Location of cursor in window 4 (bottom right).

xdif, ydif Difference between column or row of location of projectile firer and those of its target square.

xf, yf Map location *from* which a unit is being moved.

xi, yi Map location of square of impact of a projectile.

xm, ym Map location of unit (column, row). *xn, yn*. Next location of cursor in window 0 (main screen).

xn4, yn4 Next location of cursor in window 4 (bottom right).

xp, yp Screen location of the *symbol* of a unit which is being moved.

xs, ys Screen location of unit (column, row).

xw, yw Map location of a projectile-firing unit.

xz, yz Map location of a square being searched by FINDUNIT.

Arrays

aa(army number) Base addresses of army details.

aname\$(army number) Names of armies.

com(category, index) Close Combat data. Category 1/2, details are unit numbers of army 1 and 2. Category 3/4, details are combat potentials of these units of army 1 and 2. Index is *ctn1, ctn2* or *ctnr*.

d(army number, unit number) Flags to indicate if a unit has moved or fired in that turn; =1 when moved or fired.

ho(unit number) Hidden status of unit; =1 if hidden.

ob(army number) Base address of first order stack of an army.

pa(army number) Paper colours of army unit symbols.

pe(army number) Pen colours of army unit symbols.

rb(army number) Base address of first report stack of an army.

t(index) Unit numbers of enemy units present on a target square. Index is *tt*.

units(army number) The number of units in an army.

xy(category, index) Holds map locations of squares on which close combat is occurring. Category 1 = *xm*; category 2 = *ym*; index is *nxy*.

Appendix

Useful information

B

Books

Battle! — Practical Wargaming Charles Grant (Model and Allied Publications). A classic, now out-of-print, but worth searching for second-hand. Deals with World War II skirmish wargaming, with great enthusiasm.

Battles with Model Soldiers Donald Featherstone, (David and Charles). The second edition (1984) of one of the pioneering books on wargaming. A clear and concise introduction to the hobby.

Scenarios for Wargames Charles Stuart Grant (Wargames Research Group). The text is programmed to make the games suitable for one player. There are more than 20 scenarios, all of which could form the basis for a computer wargame.

PSL Guide to Wargaming Compiled and edited by Bruce Quarrie (Patrick Stephens). An excellent introduction to wargaming. The playing rules are refreshingly simple and easily adaptable to computer programs. There are good background notes for each playing period.

Games Programming Eric Solomon (Cambridge University Press). The major part of this book is directly concerned with, or is relevant to, wargame programming. It explains very clearly and in detail how to plan and write wargaming routines in any language with which you are familiar.

Battlefields of Britain David Smurthwaite (Webb and Bower, Exeter). Details of all major battles and battlefields in Britain from Roman times to World War II (including Naseby), linked by a brief commentary on the history of each period.

Great Battlefields of the World John Macdonald (Guild Publishing). Major battles of the period from Cannae (BC216) to Dien Bien Phu (AD1954), including Naseby and Omdurman. Excellent pictorial descriptions of the battlefields.

Omdurman Phillip Ziegler (Collins). A graphic account of the events

leading up to the battle and of the battle itself. Fascinating background reading for players of the OMDURMAN wargame.

Periodicals

Miniature Wargames Published monthly by Morgan Publications, available from newsagents. Plenty of scenarios, articles on tactics and other aspects of wargaming.

Military Modelling Published monthly by Argus Specialist Publications, available from newsagents. The emphasis is on modelling, but there are regular features on wargaming, including scenarios.

The Magazine Published every two months by Andy Dumelow, The Lock House, Litchfield Road, Branston, Burton-on-Trent, Staffordshire. Single copies or subscriptions from the above address. An excellent source of well-researched ideas and information for wargames.

Society

The Solo Wargamers' Association, Jeff Bayton, 63 Beckenham Road, Guildford, Surrey.

Suppliers

Commstar RS232 interface for Amstrad computers, *Nightingale* modem, connecting cable for joining two interfaces directly (Chapter 13): Pace Micro Technology, Juniper View, Allerton Road, Bradford BD15 7AG. Telephone (0274) 488211. Price of cable approximately £10.00.

Amstrad RS232 interface and V21/23 modem: Amsoft, Victoria House, PO Box 10, Sunderland, SR1 3PY. Telephone (0783) 673395.

Mail order service

For those readers who wish to purchase these games on tape cassette:

JUNGLE ATTACK	£4.95
NASEBY	£4.95
OMDURMAN	£4.95

For those readers who wish to purchase these games on disk:

JUNGLE ATTACK	£9.95
NASEBY	£9.95
OMDURMAN	£9.95

Prices include postage and packing plus VAT.

Available from Owen and Audrey Bishop, PO Box 141, Bawtry, Doncaster, S. Yorkshire, DN10 5AH.

Cheques payable to Owen and Audrey Bishop.

Delivery will normally be within 7 days but please allow 28 days.

Index

- Addresses, storage of, 47
- ADVANCE subroutine, 66–67, 81, 111
- Advancing, 62–63, 138
- Armies, 57–58, 79, 97, 101–102, 130, 143–144, 146–147, 161
- ASCII codes, 20–21
- AVDICE subroutine, 127
- AVDIE subroutine, 127
- Average dice, 126

- BATTLE subroutine, 113–114, 123
- BLOCK routine, 29–30
- BLOCKED subroutine, 73, 83
- BOOM subroutine, 67, 81
- BRIEFING subroutine, 89–93, 109–110, 113

- CARTOGRAPHER program, 9, 19–27
- Chaturanga*, 1
- Chess, 1, 4
- Cipher program, 8, 75–77, 115–119, 156–160
- CLEAR2 subroutine, 73, 83
- Close combat, 65
- CLOSE subroutine, 43–44, 69, 81
- Combat efficiency value, 39–43, 80, 121
- Combat potential, 42–43, 65
- Combat resolution, 4, 126, 131
- COMBAT subroutine, 43, 140, 155
- COMMAND subroutine, 68, 81
- Computer control, 61–64
- CONCEAL subroutine, 114, 123
- Constants, 129
- Control, *see* Computer control
- Cover, 37, 84
- Cursor, *see* Computer control
- CURS0R0 subroutine, 68, 81
- CURS0R4 subroutine, 68, 81

- Data, army, 48–49, 140–142
- Data, map, 49–50

- Data tables, 50
- Deploying units, 64
- DEPLOY subroutine, 67, 81
- Dice functions and subroutines, 126–128
- Digging in, 62, 63, 81
- DIGIN subroutine, 74, 83
- Disorganisation, 108
- DISPARMY subroutine, 67, 81, 149

- Equipment, 7, 165, 171–172

- FINDUNIT subroutine, 69, 81
- FIRE subroutine, 72, 83
- Firing, 62, 63
- Fog of war, 133–134

- Go, 1
- Graphics symbols, 20–21, 58–59, 99–100
- Grenades, 37, 62
- Group combat, 39–44

- HIDECOVER subroutine, 85
- HIDEDIST subroutine, 134, 163
- HIDEOBST subroutine, 114, 123, 155
- Hidden movement, 84–87, 98

- INITIAL subroutine, 68, 81
- INTELLIGENCE program, 9, 52–54
- Interface, 7, 165

- Key-in way, 8

- Line of sight, 86–87, 98
- LISTEN subroutine, 167, 170
- LOS subroutine, 74, 83, 86–87

- Machine code, 8
- Map, 11, 24, 49–50, 95, 145
- Map scale, 18–19, 56, 100, 147
- MCODE, 8, 17, 27–31, 45
- Mêlée, *see* Close combat
- MENDIT subroutine, 71, 83
- Memory map, 45–48
- MESSAGE subroutine, 73, 83
- Missile weapons, 32–37
- MISSION subroutine, 134, 139, 150, 154
- Modem, 7, 171
- Morale, 10, 66, 91, 107–108
- Mortar fire, 37–38, 64
- MORTAR subroutine, 73, 83
- Movement, 56–57, 105
- MOVE subroutine, 74, 83

- NDICE function, 127
- NDIE function, 126
- NEWMAP subroutine, 68, 81
- Normal dice, 126

- Operations table, 134–135
- OPSTABLE subroutine, 135
- Orders, 135–137, 140–141

- Panic, 10
- PATCH subroutine, 71, 83, 113
- PDIE function, 127
- Percentage dice, 126
- PHASE subroutine, 69, 81, 112
- Pike and shot, 95–98
- PIP subroutine, 67, 81
- Projectiles, 37–39
- PROJECTILE subroutine, 73–74, 83
- PROX routine, 30–31, 43
- PROX subroutine, 69, 81

- Rallying, 60, 103
- RANORM function, 35, 41
- RECRUITER program, 9, 12–16
- Reforming units, 108
- Reinforcing, 109, 136
- Reports, 136–137, 138–139, 141–142
- REPORT subroutine, 140, 154–155
- RESCOM subroutine, 43, 44, 70, 81
- Research, 129–130
- Rifle fire, 32–37
- Rostering, 62, 63
- Routing, 103
- Rules, 2, 61, 88, 99–103, 131, 147–148

- SAVE subroutine, 71, 82
- Saving a game, 65–66
- Scenario, 2, 56, 94–95, 124–125, 143, 146
- SETMAT routine, 30
- SEND subroutine, 167, 170
- Simultaneous combat, 42, 104
- Skirmish wargames, 42, 55
- SMAP routine, 29
- STACKS subroutine, 140, 154
- Standard deviation, 33–34
- START subroutine, 71, 82, 112, 149
- Status, 10, 49, 59, 90
- Strategy, 5
- Symbols, *see* Graphics symbols
- SYMBOLS subroutine, 74, 83, 113, 125, 150

- TABLER program, 9, 50–54
- Tactics, 5
- TALK subroutine, 167, 170
- Terrain, 2–3, 11, 17, 56–57, 146
- Time scale, 18, 100, 147
- Transmission of data, 174–176
- TURNEND subroutine, 70, 81, 149
- Two-computer programs, 166–168

- UNITLINE subroutine, 67, 81, 112, 149
- Units, Fighting, 2, 10–12, 130
- Unit scale, 18, 100, 130
- Utility way, 8, 12–16, 78–80, 120–122, 160–162

- Victory conditions, 61, 110, 131
- View phase, 164
- VIEW subroutine, 167, 169
- Visibility, 84–87

- Wargame design, 130–132
- Wargaming,
 - Board, 4–5
 - Computer, 5, 6, 7
 - Modern, 2
 - Miniature, 4–5, 7, 55
 - Origin of, 1
 - scales, 17–19, 100, 130, 146–147
 - Solo, 171
 - system, 8, 128–129
- Weapons, 60

WARGAMING ON THE AMSTRAD

CPC 464,664 & 6128

Wargames are different! The book includes three complete wargame programs ready to type in and run. These programs are also available ready-recorded on tape or disk (see Appendix). *Jungle Attack* is a skirmish wargame set on a Pacific island in World War 2. By contrast, *Naseby* re-creates the famous battle of the English Civil War between Charles II and the army of Parliament. Had Charles won, the course of English history might well have been different. Can you command the Royalist troops and lead them to victory? The two previous games include hidden movement routines to add greater realism. The third game, *Omdurman*, is set in the Sudan, the scene of Kitchener's final battle against the forces of the Mahdi. Realism is even further enhanced in this game by the independent action routines.

But this is not just a book of computer games. The early chapters explain the principles of wargaming and how the Amstrad may be used as a wargaming computer. The system upon which the wargames programs are based is described in depth. This makes it easy for readers to adapt and extend the programs to their own requirements. Several utility programs are listed, to assist readers who wish to design and implement their own wargames.

Only a simple lead is required to connect two Amstrads together for playing two-computer wargames. The way to adapt *Jungle Attack* and other programs for this novel and exciting approach to wargaming is explained in the text. Taking this one stage further, the last chapter explains how to play wargames by telephone, using a modem. For 'comms' enthusiasts, solo wargamers and others, modem wargaming is an up-to-the minute, absorbing pastime.

The publishers would like to thank Pace Micro Technology for loan of equipment used in the preparation of this book.

ISBN 0-85242-888-X



9 780852 428887

MAKING THE ADVANCED 4.6 & 6I 28 **REGIONS**

AMSTRAD CPC



MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA



<https://acpc.me/>

[FRA] Ce document a été préservé numériquement à des fins éducatives et d'études, et non commerciales.

[ENG] This document has been digitally preserved for educational and study purposes, not for commercial purposes.

[ESP] Este documento se ha conservado digitalmente con fines educativos y de estudio, no con fines comerciales.